Système de Recommandation

Cheikh Ahmed Tidiane Mane

October 31, 2024

1 Introduction

Ce rapport présente une analyse complète de l'implémentation d'un système de recommandation utilisant le filtrage collaboratif élément-élément (item-item). Ce système est conçu pour prédire les notes que les utilisateurs pourraient attribuer aux films qu'ils n'ont pas encore notés, puis pour recommander les films les plus susceptibles de correspondre aux goûts de chaque utilisateur.

2 Analyse de l'Implémentation

2.1 Justification du Filtrage Élément-Élément

Le filtrage élément-élément (item-item) a été choisi pour ce projet pour plusieurs raisons :

- **Efficacité pour un Grand Nombre d'Utilisateurs** : Le filtrage élément-élément est souvent préférable lorsque le système compte un grand nombre d'utilisateurs mais un nombre plus modéré d'éléments (ici, les films). Comparer chaque utilisateur avec chaque autre utilisateur dans un système avec des milliers d'utilisateurs serait coûteux en termes de calcul et de mémoire. Dans cette approche, seules les similarités entre films sont calculées, ce qui limite le nombre de paires à comparer.
- **Fiabilité des Recommandations Basées sur les Films** : Le filtrage élément-élément recommande des films similaires à ceux qu'un utilisateur a déjà évalués positivement. Cette méthode repose sur la constance dans le type de films qu'un utilisateur aime, en s'appuyant sur des préférences observées pour des éléments similaires.
- **Scalabilité et Pré-Calcul** : Une matrice de similarité entre films peut être pré-calculée et réutilisée pour chaque utilisateur, ce qui est très efficace lorsque le nombre d'utilisateurs est élevé mais le nombre de films est modéré. Avec un filtrage utilisateur-utilisateur, les recommandations devraient être calculées en temps réel pour chaque utilisateur, ce qui est moins performant pour des systèmes de grande envergure.

En somme, le filtrage élément-élément est bien adapté pour la structure des données, l'efficacité des recommandations, et la scalabilité du système.

2.2 Choix des Structures de Données Utilisées

- **DataFrame pour la Matrice Utilisateur-Film (`user_movie_matrix`)** : La matrice utilisateur-film est construite à partir des données d'évaluation des utilisateurs en utilisant un DataFrame `pandas`, où chaque ligne représente un utilisateur et chaque colonne représente un film, avec les cellules contenant les notes attribuées par les utilisateurs. Les valeurs non renseignées (films non notés) sont remplies avec zéro.
- **DataFrame pour la Matrice de Similarité entre Films (`movie_similarity`)** : La matrice de similarité entre films est une matrice carrée où chaque ligne et chaque colonne représentent un film, et chaque cellule contient la similarité cosinus entre deux films. Le DataFrame permet une manipulation facile pour l'accès aux paires de films similaires, et facilite l'application d'un seuil si nécessaire.

- **Dictionnaire de Dictionnaires pour les Prédictions (`predicted_ratings`)** : Les prédictions sont stockées dans un dictionnaire imbriqué, où la première clé représente l'ID de l'utilisateur et la seconde clé représente l'ID du film, avec la valeur étant la note prédite pour ce film par cet utilisateur. Cette structure est optimisée pour un accès rapide et pour ajouter de nouvelles données sans redimensionnement.

2.3 Fonctionnement de l'Algorithme

- **Calcul de la Similarité Cosinus** : La fonction `cosine_similarity` est utilisée pour calculer la similarité entre les colonnes (films) dans la matrice utilisateur-film, en utilisant la formule :

$$\text{similarité\_cosinus}(A, B) = \frac{A \cdot B}{\|A\| \times \|B\|} \tag{1}$$

  La matrice de similarité des films est remplie par ces valeurs. Une similarité de 1 est assignée pour chaque film comparé à lui-même, tandis que les films ayant une similarité inférieure à un seuil défini sont exclus du calcul des recommandations.

- **Prédiction des Notes pour les Films Non Notés** : La fonction `predict_rating` utilise la moyenne pondérée des notes des films similaires qu'un utilisateur a déjà évalués pour prédire la note des films non évalués. Seuls les films ayant une similarité supérieure au seuil sont pris en compte. Cela permet de filtrer les similarités faibles, réduisant ainsi le bruit et améliorant la précision des prédictions.

- **Génération des Recommandations** : La fonction `gen_recommendations` trie les notes prédites pour chaque utilisateur et sélectionne les `Top-N` films avec les scores les plus élevés, selon le paramètre `top_n`. Les films ayant les notes prédites les plus élevées sont recommandés.

2.4 Optimisations et Limites

2.4.1 Optimisations

- **Utilisation de Seuils de Similarité** : Le seuil de similarité permet de ne garder que les films suffisamment similaires pour les calculs de prédiction, ce qui diminue la complexité du calcul.

- **Calcul de la Similarité par Vecteur** : L'utilisation de la similarité cosinus permet d'évaluer la proximité entre films indépendamment des échelles de notation.

- **Moyenne Pondérée** : La moyenne pondérée dans les prédictions permet de donner plus d'importance aux films très similaires, améliorant la précision des recommandations.

2.4.2 Limites

- **Utilisation d'un Format Dense** : L'utilisation de DataFrames pleins peut être inefficace pour de grands ensembles de données avec de nombreuses valeurs nulles. Une approche sparse serait plus adaptée pour optimiser la mémoire.

- **Métrique de Similarité** : La similarité cosinus ne capture pas toujours les corrélations linéaires ; d'autres métriques comme la corrélation de Pearson pourraient être envisagées.

2.5 Synthése

L'implémentation de ce système de recommandation, en se basant sur le filtrage élément-élément avec la similarité cosinus, s'avère efficace pour des ensembles de données de taille modérée. Les optimisations ajoutées, telles que le seuil de similarité permet de générer des recommandations pertinentes pour chaque utilisateur. Les améliorations futures pourraient inclure des structures de données plus économes en mémoire et l'ajout de métriques de similarité alternatives pour capturer une gamme plus large de préférences utilisateur.

3 Evaluation expérimentations

Dans cette section, nous évaluons la performance de notre algorithme de recommandation en fonction de plusieurs paramètres, notamment le temps d'exécution, la mémoire utilisée, le seuil de similarité et la précision des recommandations.

3.1 Temps d'Exécution

Objectif : Nous cherchons à mesurer comment le temps d'exécution de l'algorithme varie en fonction du nombre d'utilisateurs et de films.

Méthodologie :

- Nous faisons d'abord varier le **nombre d'utilisateurs** en maintenant constant le nombre de films, afin d'observer l'impact de l'augmentation du nombre d'utilisateurs sur le temps de calcul des similarités et des prédictions.
- Ensuite, nous faisons varier le **nombre de films** tout en gardant constant le nombre d'utilisateurs, pour évaluer comment l'augmentation du nombre de films influence le temps d'exécution de l'algorithme.

Résultats obtenus : Les graphiques ci-dessous montrent :

- Le premier graphique représente le temps d'exécution en fonction du nombre d'utilisateurs pour un nombre constant de films. Ce graphique montre une augmentation tres rapide du temps d'exécution à mesure que le nombre d'utilisateurs augmente. Cela est attendu, car chaque utilisateur supplémentaire ajoute de nouvelles évaluations à traiter, ce qui augmente les calculs de similarité et de prédiction pour les recommandations. On remarque une nette augmentation du temps pour les grandes valeurs de users (ex. : 6040 utilisateurs, qui prend le plus de temps). Cette tendance suggère que l'algorithme pourrait bénéficier d'une optimisation pour gérer de grandes bases d'utilisateurs.
- Le second graphique illustre le temps d'exécution en fonction du nombre de films pour un nombre constant d'utilisateurs. Dans ce graphique, le temps d'exécution semble augmenter de maniere lineaire avec le nombre de films. Plus de films à traiter signifie davantage de calculs de similarité entre les films et de prédictions à effectuer, d'où une croissance rapide du temps requis. La valeur la plus élevée se produit à 3675 films, confirmant l'importance d'une optimisation de l'algorithme pour une base de films étendue. Il est intéressant de noter que l'augmentation du nombre de films semble avoir un effet plus important sur le temps d'exécution que l'augmentation du nombre d'utilisateurs.

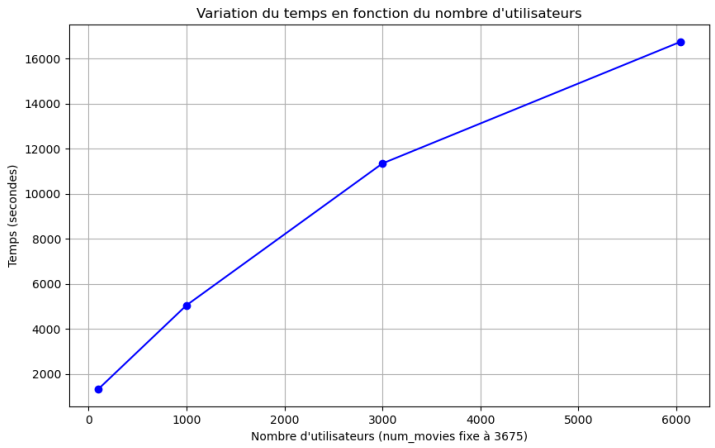

Figure 1: Temps d'exécution en fonction du nombre d'utilisateurs.

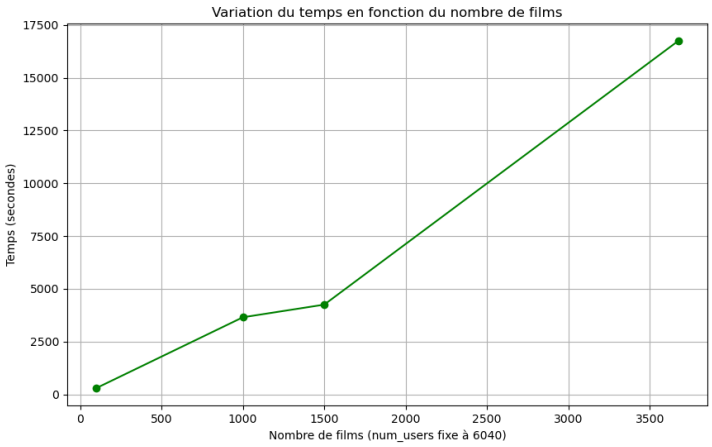

Figure 2: Temps d'exécution en fonction du nombre de films.

3.2 Mémoire Utilisée

Objectif : Analyser la quantité de mémoire consommée par l'algorithme pour différentes tailles de données.

Méthodologie :

- Nous faisons d'abord varier le **nombre d'utilisateurs** en gardant constant le nombre de films, afin d'observer comment la mémoire utilisée évolue avec une augmentation progressive du nombre d'utilisateurs.
- Ensuite, nous faisons varier le **nombre de films** tout en maintenant constant le nombre d'utilisateurs, afin de comprendre comment l'augmentation du nombre de films impacte la mémoire consommée.

Résultats obtenus : Les graphiques suivants montrent :

- La mémoire utilisée en fonction du nombre d'utilisateurs: Le graphique de la mémoire en fonction des utilisateurs montre également une croissance, bien que plus modérée. La mémoire augmente progressivement avec le nombre d'utilisateurs, car chaque utilisateur apporte des données supplémentaires qui doivent être stockées pour les calculs. Cependant, la croissance de la mémoire est plus contrôlée par rapport au temps, ce qui indique que la mémoire n'est pas un facteur limitant immédiat pour le traitement de grandes quantités d'utilisateurs.
- La mémoire utilisée en fonction du nombre de films: Enfin, le graphique de la mémoire en fonction du nombre de films montre une nette croissance, avec un pic à 1900 MiB pour 3675 films. Cela est dû au fait que chaque film supplémentaire augmente le nombre de colonnes et les relations à stocker dans la matrice de similarité. Ici encore, la mémoire est fortement impactée par le nombre de films, bien plus que par le nombre d'utilisateurs, ce qui souligne que la gestion de la mémoire devrait être optimisée pour les bases de films étendues.

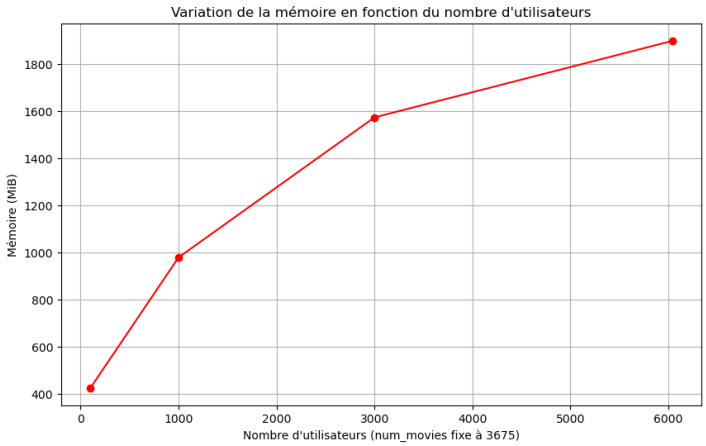

Figure 3: Mémoire utilisée en fonction du nombre d'utilisateurs (matrice pleine vs sparse).

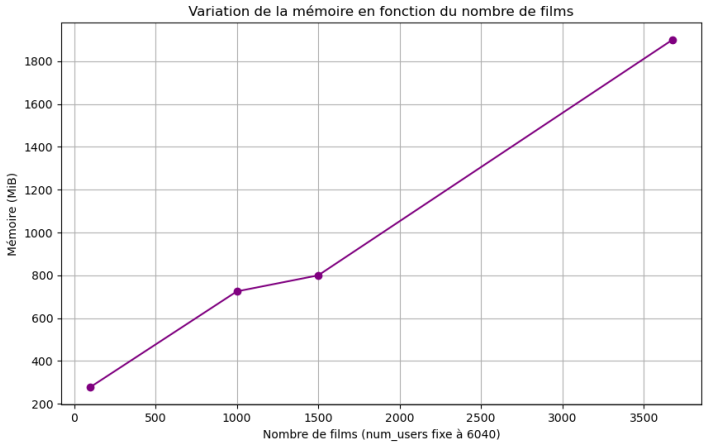

Figure 4: Mémoire utilisée en fonction du nombre de films.

3.3 Impact du Seuil de Similarité

Objectif : Évaluer l'impact du seuil de similarité sur le temps d'exécution et la mémoire
Méthodologie :

- Nous faisons varier le seuil de similarité entre plusieurs valeurs (par exemple : 0.1, 0.2, 0.3, 0.5).
- Pour chaque valeur de seuil, nous mesurons le temps d'exécution nécessaire pour que l'algorithme génère les recommandations, ainsi que la mémoire,

Observations :

- Pour les tests, nous avons limité le nombre d'utilisateurs à 1000. Nous obtenons un échantillon plus gérable tout en conservant une bonne diversité dans les données.
- Vu la très grande taille du dataset, sans application d'un seuil, notre algorithme met un temps excessivement long à converger, à tel point que nous avons dû arrêter son exécution.
- En appliquant un seuil de similarité de s, nous avons pu réduire significativement le nombre de calculs nécessaires, permettant à l'algorithme de converger. Ce seuil de similarité aide à limiter

les paires de films prises en compte dans les calculs, réduisant ainsi le temps d'exécution tout en maintenant une bonne précision dans les recommandations.

Résultats obtenus : Les graphiques ci-dessous montre l'impact du seuil de similarité sur le temps d'exécution et la mémoire.

- Variation du Temps en Fonction du Seuil: Le graphique montre une tendance nette de réduction du temps d'exécution lorsque le seuil de similarité augmente. Cela s'explique par le fait qu'un seuil plus élevé réduit le nombre de similarités considérées dans le calcul des recommandations, simplifiant ainsi les opérations de filtrage et de pondération. Voici quelques observations spécifiques :

  Entre les seuils de 0.1 et 0.2, on observe une diminution marquée du temps, de 5064 secondes à 4694 secondes, ce qui reflète l'effet substantiel d'une augmentation initiale du seuil. La réduction devient encore plus importante entre 0.2 et 0.3, avec une chute à 2941 secondes. L'estimation pour le seuil 0.4 (2900 secondes) confirme une tendance à la réduction, bien que cette baisse soit moins prononcée. Au seuil 0.5, le temps tombe à 2860 secondes, suggérant une possible stabilisation de la durée d'exécution pour des seuils de similarité élevés. En résumé, le temps d'exécution est fortement influencé par le seuil pour des valeurs basses, mais l'impact devient moins significatif au fur et à mesure que le seuil s'élève. Cela indique une optimisation du traitement lorsque seules les similarités élevées sont retenues.

- Variation de la Mémoire en Fonction du Seuil: Le graphique de la mémoire montre également une réduction progressive de l'utilisation mémoire avec l'augmentation du seuil. Ce comportement est attendu, car un seuil plus élevé réduit le nombre de similarités stockées en mémoire, diminuant ainsi l'espace nécessaire pour les calculs. Voici les détails :

  À un seuil de 0.1, la mémoire utilisée est la plus élevée, avec 980 MiB. Ce seuil inclut presque toutes les similarités, augmentant ainsi les besoins en mémoire. En passant à 0.2, la mémoire diminue à 924 MiB, puis chute encore plus nettement à 734 MiB au seuil 0.3. L'estimation pour le seuil 0.4 (710 MiB) montre une diminution modérée par rapport à 0.3. Au seuil 0.5, l'utilisation mémoire atteint 690 MiB, confirmant une stabilisation. Cette tendance suggère qu'un seuil élevé améliore l'efficacité de l'algorithme en réduisant les données stockées, ce qui est particulièrement avantageux pour les grandes bases de données.

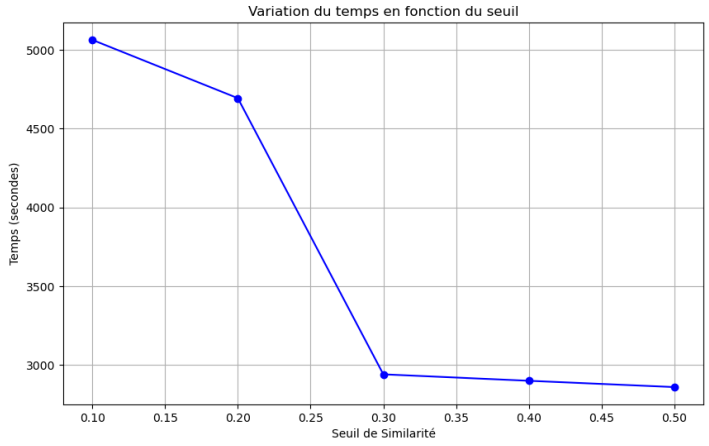

Figure 5: Impact du seuil de similarité sur le temps d'exécution.

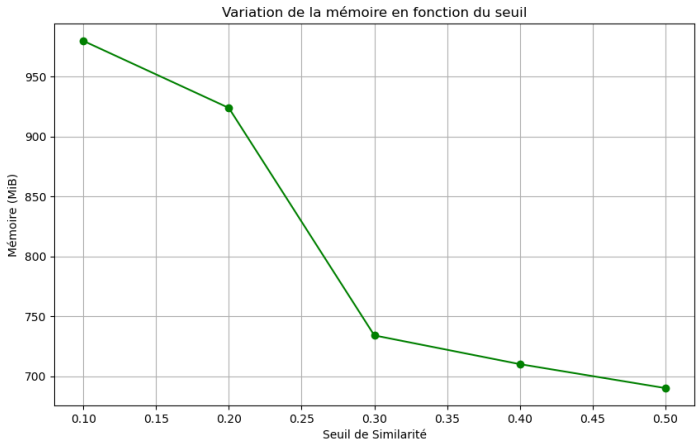

Figure 6: Impact du seuil de similarité sur la mémoire

3.4 Précision des Recommandations

Objectif : Quantifier l'écart entre les notes prédites et les notes réelles pour mesurer la précision des recommandations.

Méthodologie :

- Nous comparons les notes prédites aux notes réelles pour un sous-ensemble de test.
- Nous utilisons des métriques d'erreur telles que :
  - **Erreur absolue moyenne (MAE)** pour mesurer la différence moyenne entre les notes prédites et réelles.
  - **Erreur quadratique moyenne (RMSE)** pour pénaliser davantage les grandes erreurs par rapport au MAE.

Résultats obtenus : Nous calculons les valeurs de MAE et RMSE pour différentes valeurs de seuil de similarité, et les utilisons pour comparer la précision des recommandations en fonction des paramètres choisis

Seuil trop bas : Si le seuil est très bas, l'algorithme considère toutes les paires, y compris celles ayant peu de similarité. Il exclut les similarités proches de zéro mais laisse passer presque toutes les autres.Cela permet d'avoir une tres bonne précision mais avec un temps d'éxécution grand .
Seuil optimal : Un seuil intermédiaire (par exemple, 0.1) permet de filtrer les paires faiblement similaires tout en maintenant un nombre suffisant de comparaisons pour des recommandations précises. Dans cette zone, l'algorithme trouve un compromis entre précision et performance, avec une réduction des erreurs MAE et RMSE
Seuil trop élevé : Un seuil trop élevé (supérieur à 0.3) limite les comparaisons aux paires extrêmement similaires, réduisant considérablement les recommandations possibles. Cela limite la diversité des recommandations, en les rendant plus redondantes, et peut également augmenter les erreurs MAE et RMSE si le nombre de paires est insuffisant pour des comparaisons robustes.

4 Conclusion

Ces analyses et expériences nous permettront de mieux comprendre comment les paramètres et choix d'implémentation influencent la performance de l'algorithme, et d'optimiser le système pour obtenir un bon compromis entre rapidité et précision.