Projet de Data Science

Réalisé par: BENNISS Salma
HADJKACEM Hibatallah
GAO Tianjinzi
TOUZI Marwen

Plan

- Introduction
- Statistique descriptive
- Modélisation:
  - Memory Based
  - Model Base
- Solution Finale
- Conclusion

Introduction:

- L'objectif de ce projet était la création d'un système de recommandation performant et scalable. On a mesuré la performance avec les indice d'évaluation tels que le RMSE (Root mean square error) et la scalabilité par la possibilité d'appliquer nos algorithmes sur un jeu de données de 10 millions d'observations.
- Pour le faire, on a utilisé des modèles qui se basent sur la similarité (Memory Based) et des modèles qui se basent sur la factorisation des matrices (Model Based).
- On a appliqué ces algorithmes sur une partie de notre jeu de données, mais en montant dans les dimensions, Python n'a pas accepté une matrice de 7636 lignes et 1264 colonnes.
- Notre solution était de passer des matrices aux tuples. Une solution qui a été efficace et qui retourne des résultats avec un bon temps d'exécution et un petit taux d'erreur.

Statistique Descriptive:

- La taille de données: **10 000 054** lignes, **69 878** users, **10 677** items
- Fréquence:

item	nbr	rates	min_rate	max_rate
110	39	0.39	2.5	5.0
527	38	0.38	2.0	5.0
296	38	0.38	1.0	5.0
593	33	0.33	1.0	5.0
590	33	0.33	1.0	5.0

- Max et Min des ratings pour chaque item

- **Séparer le train et le test par la date de ratings ?**

1. Il n'y a **pas d'item** qui a été **noté 2 fois** par un meme **User**
2. Pour un user, les dates de Rating sont proches. Parmi **69 878** users, il y a **8 323** users **[11.91%]** dont les années de ratings sont différents.

UserID	ItemID	Rating	Date_rating
1	122	5	**1996-08-02**
1	185	5	**1996-08-02**
...	...	...	...

Modélisation:

- Il existe deux type de système de recommandation:
  - -Le Content Based: qui se base sur les caractéristiques des users/items (on va pas l'utiliser vu qu'on n'a pas ces données.)
  - Le Collaboratif Filtering: qui se base sur seulement la note attribué par l'utilisateur sur l'item.

- Il existe deux type de recommandation Collaboratif Filtering:
  - Memory Based
  - Model Based

CF: Memory based:

- Les modèles Memory based se base sur la similarité calculé par les différentes métriques d'évaluation. Pour nous, on a utilisé le Cosine similar et le citybloc.
- Il existe deux type de Memory Model:
  - User-Item based : cet algorithme va prendre un utilisateur, trouve les utilisateur les plus similaires à lui, Puis recommande les items aimé par ces utilisateurs (ça prend un user et retourne des items). En général on l'utilise pour dire "Les utilisateurs qui sont similaires à vous, ont aimé aussi ..."
  - Item-Based : prend un item, cherche les utilisateurs qui ont aimé cet item et recommande les items aimé par ces utilisateurs (ça prend un item et retourne une liste des items). En général on l'utilise pour dire "Les utilisateurs qui ont aimé ça, ont aimé aussi ..."
- Voilà le RMSE qu'on a trouvé:

Modèles:	User-Based "Cosine"	Item-Based "Cosine"	User-Based "Citybloc"	Item-Based "Citybloc"
RMSE:	1.498	1.517	1.498	1.460

CF : Model Based:

- Les Model Based se basent sur la multiplication de matrices ou **Matrix Factorisation (MF) :** Méthode d'apprentissage non supervisé de décomposition et de réduction de dimensionnalité en utilisant les caractères latents (k : espace des caractères cachés)

- Le but de ces algorithmes est de trouver les **préférences latentes des utilisateurs** (sous forme d'une **matrice P**) et des **items** (sous forme d'une **matrice Q**) à partir des notes attribués. Puis, la **multiplication** de ces deux matrices de structure de faible rang donne la **matrice de prédiction.**

- Une fois c'est fait, selon la méthode qu'on va utiliser, on améliore P et Q pour avoir un produit qui est conforme à la **matrice des notes originale**. Voilà les différentes méthodes qu'on a utilisé:
  - **SVD Singular Vector Decomposition: X=U x S x V.T** pour le faire on a utilisé la librairy **scipy.sparse.linalg**. **U** présente la matrice **User-Latent** et **V.T** la matrice I**tem-Latent**
  - **NMF Non-Negative Matrix Factorisation**: On construit le modèle, on lui fait l'apprentissage avec la base train (pour obtenir P et Q). On a utilisé cet algorithme avec la library **sklearn.decomposition**

CF : Model Based:

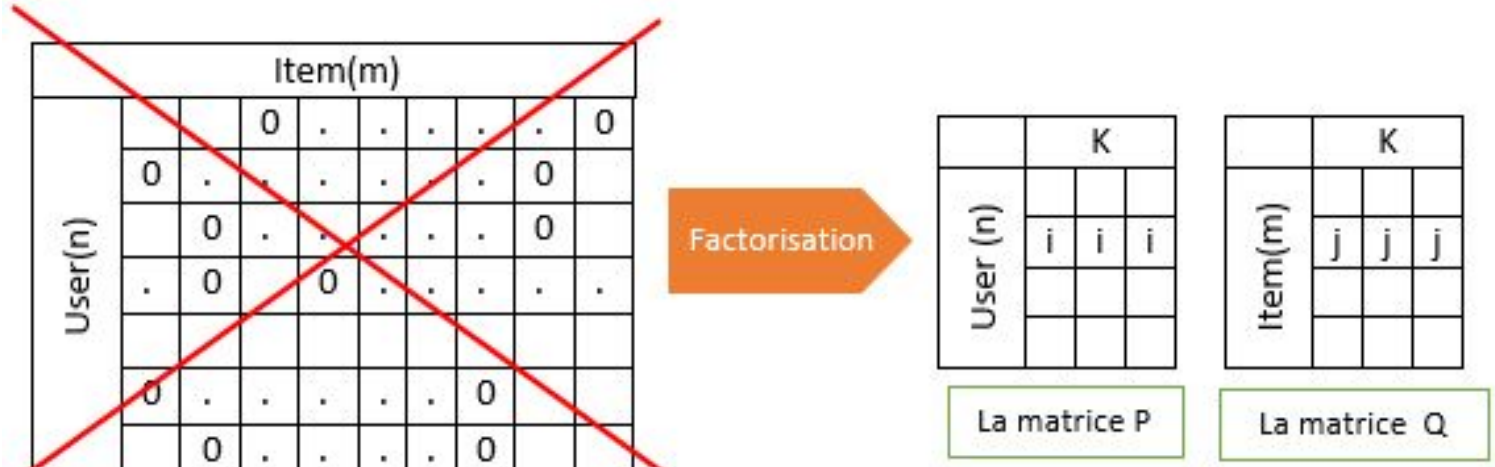
- **ALS Alternating Least Squares & SGD Stochastic Gradient Descent** : qui présentent deux manières pour minimiser l'erreur de régularisation carré. Le principe est de prendre aléatoirement P et Q et appliquant selon le cas une formule mathématique pour rendre le produit le plus proche de la matrice originale. Voilà la **différences entre ALS et SGD:**

Algorithmes :	Les avantages:	Les désavantages
ALS	- On peut l'utiliser en // -Plus performant dans un cas non sparse.	-En général, il est plus lent
SGD	-Plus facile à implémenter et plus rapide.	-Lent dans le cas d'une matrice sparse.

- Voilà les **résultats** que nous avons obtenu:

Modèles:	**SVD**	**NMF**	**ALS**	**SGD**
RMSE:	1.490	1.73	1.13	1.18

Solution Finale: Matrice de Factorisation



UserID	ItemID	Ratings
i	j	R(ij)

Méthode de descente du gradient stochastique :

$$e_{ui} = r_{ui} - q_i^T p_u$$

$$q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i)$$

$$p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda \cdot p_u)$$

```
train RMSE: 1.025081, validation RMSE: 1.025433
```

⇒ Produit à recommander pour l'User 19997: [1069 2951 10583 195]

Conclusion:

- Pour résoudre notre problème on a utilisé tous les modèles qu'on a trouvé pour comparer et valider les résultats théoriques.
- Pour les Memory Based:
  - les modèles sont facile à implémenter, ils génèrent des bonnes résultats mais ils ne résolvent pas les problèmes de Cold Start(pas d'informations sur les users & items), de sparsity(rareté des données) et de scalability (monter dans les dimensions).
- Pour les Model Based:
  - Ils ont résolu le problème de sparsity. Et chaque algorithme a ces points faibles et ces points forts.
- Pour l'utilisation de la matrice de factorisation finale:
  - On a résolu le problème de scalability et de sparsity.
  - Mais on a pas exploité la variable date_ratings pour que nos tests soient plus significatifs

Fin de la présentation

Merci pour votre attention