Projet : Matrix factorization for recommender systems

- UE: Projet science des données-

Boulif Kaoutar
Elghourbal Salma
Eric N'guessan

I- Introduction

Ce document résume notre travail effectué sur la factorisation de matrice dans le but du filtrage collaboratif. Nous commencerons d'abord par présenter le problème et faire une étude préliminaire des données fournies pour cette étude. Ensuite nous présenterons les différentes méthodes d'apprentissage auxquelles nous avons opté avec le résultat de leurs expérimentations. Enfin nous comparerons les résultats des différentes approches sur deux jeux de données, un petit et un plus grand.

II- Formulation du problème

Le filtrage collaboratif (collaborative filtering) est sous-jacent aux systèmes de recommandation. C'est l'ensemble des techniques qui visent à opérer une sélection sur les éléments à présenter aux utilisateurs, qui correspond à l'étape de filtrage en prenant comme référence le comportement et les goûts exprimés par d'autres utilisateurs dans le cadre de la collaboration. Il existe différents types de filtrage collaboratif. Nous nous intéressons dans cette étude au filtrage collaboratif basé sur la technique de factorisation de matrice. En particulier, nous traitons les algorithmes suivants: Alternating Least Squares (ALS), Descente de Gradient (GD), Probabilistic matrix factorization (PMF) et la décomposition en valeurs singulières (SVD).

III- Exploration des données

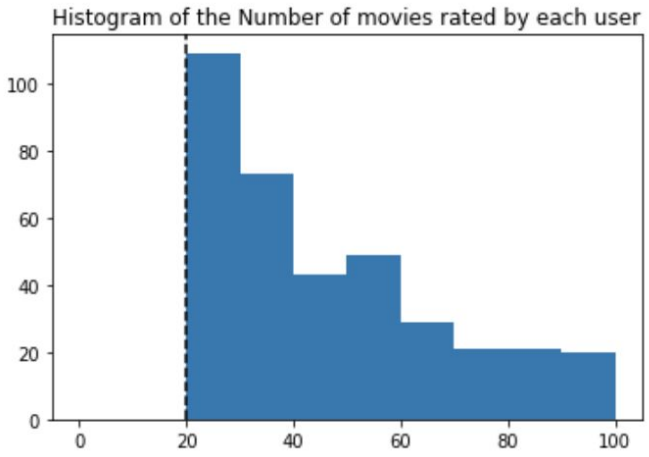
1- Compréhension des données

Pour cette étude, nous utilisons la base de données MovieLens qui décrit les préférences exprimées par les gens pour des films. Ces préférences sont sous la forme de tuples (user, item, rating, timestamp). Une personne exprime une préférence (une note de 0.5 à 5 étoiles) pour un film à un moment donné.
Pour des raisons de simplicité, nous choisissons de travailler avec la petite base de données "Small" qui contient 100,000 ratings, environ 9000 films et 600 utilisateurs. Cette base de données est disponible sur le site : http://files.grouplens.org/datasets/movielens/ml-latest-small-README.html

Nous avons pu construire la matrice des notes R qui contient pour chaque utilisateur les notes qu'il a attribuées aux films qu'il a regardés. Nous avons également réalisé des statistiques sur les données :

- **La matrice est à 98% creuse .**
- **Chaque utilisateur a au moins noté 20 films**
- La majorité des films ont moins de 25 notes.

La figure suivante illustre la deuxième observation



2- Préparation des ensembles d'entraînement et de test

Le partage des données en données d'apprentissage et données de test n'est pas a priori évident dans un problème de factorisation de matrice puisque les données ne peuvent pas être séparées comme dans un problème classique d'apprentissage automatique. En prenant en compte l'observation faite précédemment à savoir que chaque utilisateur a au moins noté 20 films, nous avons proposé une méthode de "split" en train/test comme suit :

- **Train set =** la matrice R pour laquelle nous enlevons aléatoirement 15 notes par utilisateur.

- **Test set =** une matrice de même dimension contenant uniquement les 15 notes par utilisateur qu'on a enlevées.

Ainsi, avec cette stratégie, on obtient à peu près 90% d'exemples (triplet user, movie, rating) dans l'ensemble d'entraînement et 10% d'exemples dans l'ensemble de test.

Dans tout ce qui suit, nous procédons de la manière suivante :

1. Entraîner le modèle d'apprentissage sur la matrice de train en définissant une fonction de coût à minimiser selon le type d'algorithme.
2. Obtenir la matrice de prédiction $\widehat{R}$ qui permet d'approcher la matrice R_{train} mais aussi qui permet de prédire les valeurs non nulles de R_{test}
3. Calculer une RMSE (root mean square error) entre les entrées non nulles de R et de R_{train} d'une part et d'autre part une RMSE entre les entrées non nulles de R et de R_{test}. Ces deux métriques d'évaluation sont utiles pour comparer les performances des différentes approches implémentées.

IV- Méthodes d'apprentissage :

Le but dans les deux méthodes suivantes est de trouver les deux matrices I et U qui minimisent la fonction de coût qui s'exprime de cette façon :

$$C(I,\ U)\ =\ \|R\ -\ IU^T\|^2_F\ +\ \lambda\|I\|^2_F\ +\ \mu\|U\|_F{}^2 \quad (1)$$

Où $\|.\|_F$ représente la norme de Frobenius, λ et μ des facteurs de normalisation. L'expression devient :

$$C(I,\ U)\ =\ tr(R^TR)\ -\ 2tr(R^TIU^T)\ +\ tr(UI^TIU^T)\ +\ \lambda tr(U^TU)\ +\ \mu tr(I^TI) \quad (2)$$

Pour se faire, nous dérivons la fonction de coût par rapport à I et à U. On a donc les dérivées suivantes :

$$\frac{\partial C}{\partial U}(I,U)\ =\ -2R^TI\ +\ 2UI^TI\ +\ 2\mu U \quad (3)$$

$$\frac{\partial C}{\partial I}(I,U)\ =\ -2RU\ +\ 2IU^TU\ +\ 2\lambda U \quad (4)$$

Le but sera donc d'utiliser ces dérivées pour trouver un optimum et ainsi trouver les bonnes matrices I et U résumant au mieux la matrice R contenant les notes des utilisateurs par rapport aux différents films.

a- Alternating Least Squares (ALS)

a- 1. Méthodologie

Considérons le problème de minimisation suivant:

$$min_{I,U}C(I,U) \quad (5)$$

Ce problème n'est pas convexe. Toutefois, si on fixe une des variables soit I ou U, alors le problème devient quadratique et nous pouvons résoudre la variable restante non fixée comme un problème des moindres carrés. Procéder ainsi en alternance est la principale idée de l'algorithm ALS (Alternating Least Squares).

Concrètement, voici le pseudocode de l'algorithme:

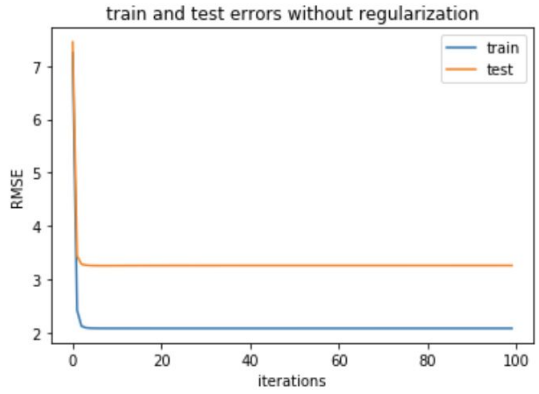
1. Initialiser les matrices I et U aléatoirement
2. Répéter à chaque t TANT QUE la condition d'arrêt n'est pas satisfaite:
   - $I_{t+1} = RU_t(U_t^TU_t + \lambda I)^{-1}$
   - $U_{t+1} = R^TI_t(I_t^TI_t + \mu I)^{-1}$

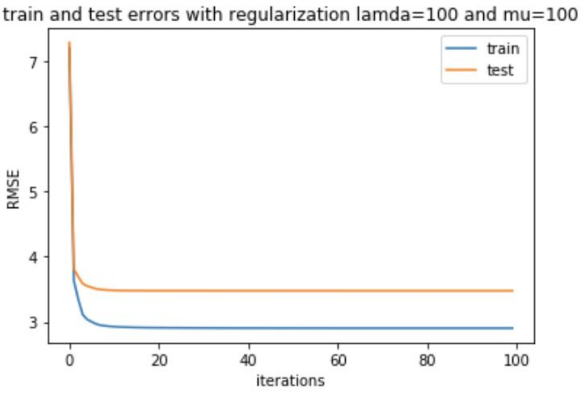
En appliquant ces étapes en alternance aux matrices I et U, nous pouvons améliorer itérativement la factorisation matricielle.

a- 2. Implémentation

i. Effet de la régularisation sur RMSE

La première expérience est de voir l'effet de la régularisation sur la RMSE du training set et du test set.

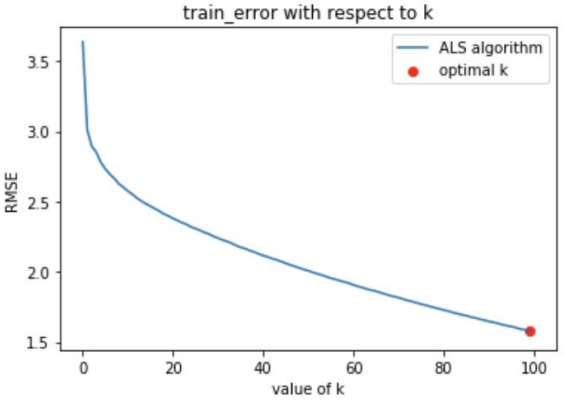


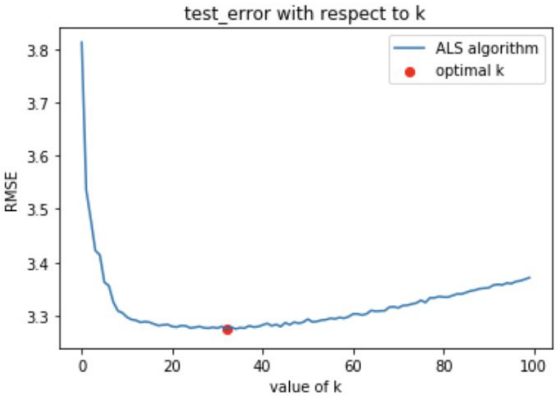


D'après les figures ci-dessus, la régularisation peut réduire l'écart entre la RMSE du training et du test set, mais ne réduit pas les valeurs de RMSE. Toutefois, ce qui est remarquable pour l'algorithme RMSE est sa convergence rapide au bout de moins de 10 itérations dans notre cas.

ii. Effet de la variable latente k sur RMSE

L'expérience suivante consiste à faire varier la valeur de la variable latente k de 1 à 100 et de voir son impact sur l'évolution de le RMSE. Les autres paramètres (λ, μ) étant fixés.





On peut voir des deux courbes que le RMSE diminue en augmentant la valeur de k pour le training set. Sur le test set, il existe une valeur k optimale qui est aux alentours de 32, elle permet d'atteindre un RMSE de l'ordre de 3.28.

b- Descente de Gradient (GD)

b- 1. Méthodologie

On va ici utiliser l'algorithme de la descente de gradient qui se déroule comme suit :

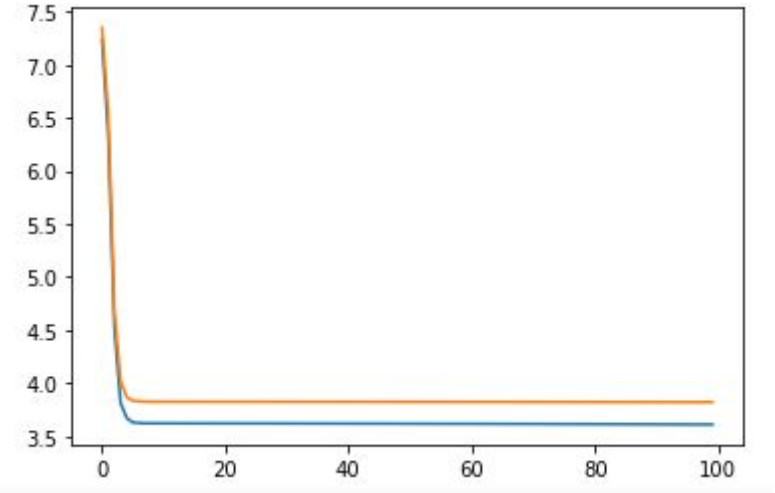
1. On initialise les matrices I et U
2. On répète à chaque t TANT QUE $C(I,\ U)\ \geq \varepsilon$:
   - $I_{t+1}\ =\ I_t - \eta_t \frac{\partial C}{\partial I}(I_t, U_t)$
   - $U_{t+1}\ =\ U_t - \psi_t \frac{\partial C}{\partial U}(I_t, U_t)$

On trouve alors les 2 matrices I et U minimisant la fonction de coût au mieux et donc approchant au mieux la matrice R.

b- 2. Implémentation

i. Evolution du RMSE à pas constant

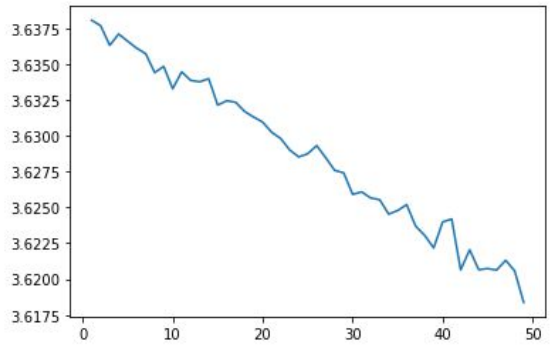
En faisant évoluer à pas constant de **0.0001**, avec $\lambda\ =\ \mu\ =\ 1$, et un $k\ =\ 40$ on obtient comme RMSE :

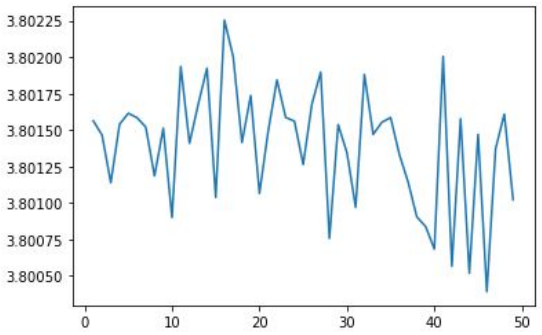


L'algorithme converge au bout de quelques itérations, sur le *training set* (en bleu) comme sur le *test set* (en orange). Le RMSE sur le training set est moins élevé que sur le test set, ce qui témoigne d'un overfitting.

ii. Evolution du RMSE lorsqu'on fait varier la dimension latente k

On décide de faire varier la dimension latente de 1 à 50 pour évaluer leur effet. On regarde les résultats sur le *training set* et le *test set*. On obtient les résultats suivants (à gauche les résultats sur le *training set*, à droite ceux du *test set*).





La dimension latente k influe donc sur le RMSE. On remarque que plus k augmente, plus l'apprentissage est meilleur sur le training set. Cependant dans le cas du test set, le k optimal se situe au niveau de 46, un ajout entraînerait une erreur plus grande.

c- Probabilistic matrix factorization (PMF)

c-1. Aperçu de l'approche

La factorisation probabiliste de matrices (PMF) permet de résoudre le même problème formulé qu'avant qui est de trouver les matrices U et V telles que : $R \simeq UV^T$. La particularité de cette méthode est qu'elle possède une interprétation probabiliste. En effet, il s'agit d'un problème d'estimation des paramètres $U et V$ à partir de la distribution des données R qui peut se résoudre en supposant que les distributions sont gaussiennes et en utilisant la propriété de Bayes :

$P(U, V \mid R, \sigma^2) = P(R|U, V, \sigma^2)P(U|\sigma_U^2)P(V|\sigma_V^2)$ (*)

L'objectif est de maximiser la probabilité A posteriori $P(U, V \mid R, \sigma^2)$. Avec un peu de calcul et en introduisant des logarithmes sur l'équation (*), on trouve que cela revient à minimiser l'inverse de fonction objective :

$$L = -\frac{1}{2}\left(\sum_{i=1}^{N}\sum_{j=1}^{M}(R_{ij} - U_i^T V_j)^2_{(i,j)\in\Omega_{R_{ij}}} + \lambda_U \sum_{i=1}^{N}\|U_i\|^2_{Fro} + \lambda_V \sum_{j=1}^{M}\|V_j\|^2_{Fro}\right)$$

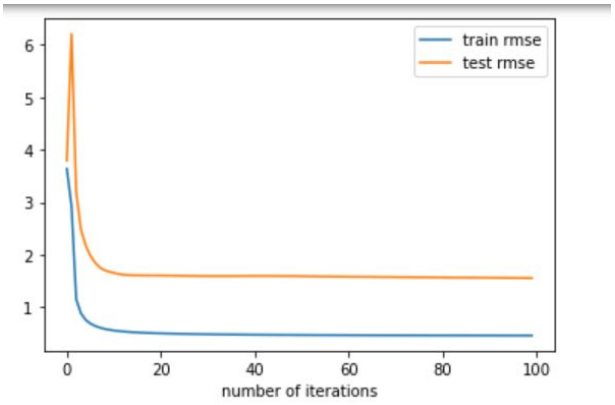
avec :

- L est appelé le Log a posteriori
- Rij est l'élément d'indice (i,j) de la matrice R.
- Ω_{Rij} est l'ensemble des ratings Rij observés dans R
- λ_U, λ_V des constantes de régularisation.
- Ui est la ligne i de U.
- Vj est la ligne j de V.
- N est le nombre de users et M est le nombre de films.
- On utilise la norme de Frobenius.

La minimisation de l'inverse de L se fait en mettant à 0 ses dérivés par rapport à Ui et Vj et en utilisant une ALS sur chaque Ui et Vj jusqu'à convergence. L'article [1] contient l'algorithme complet de la PMF.

c-2. Résultats de l'implémentation

i. Evolution de l'erreur RMSE sur le train et sur le test en fonction du nombre d'itérations :

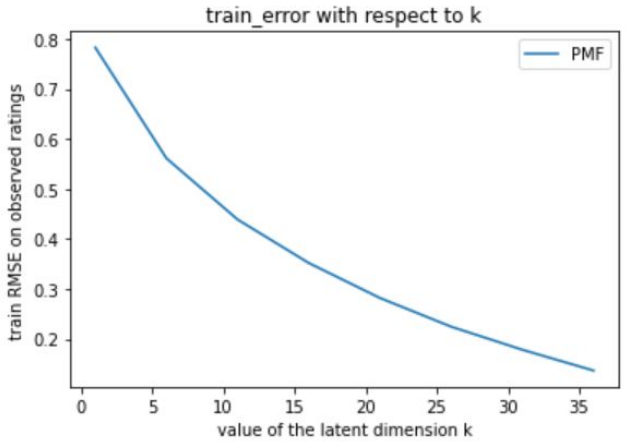


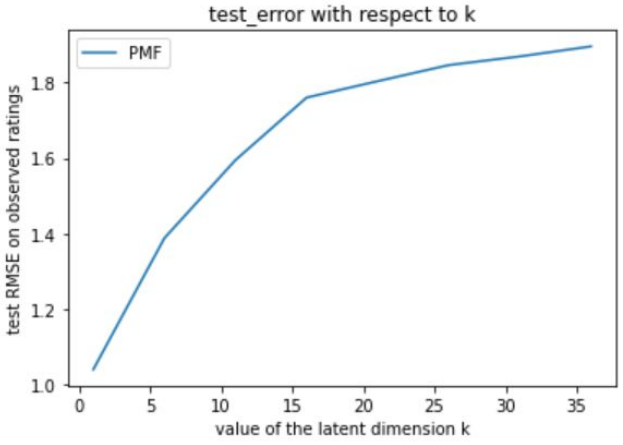
PMF with hyperp : n_dims = 10, lambda_U = 0.1, lambda_V = 0.1, n_iter = 100

L'algorithme converge au bout de 10 itérations, ce qui est très rapide. L'erreur sur le train converge vers une valeur faible mais l'erreur de test converge vers une valeur un peu plus élevée. Nous remarquons un léger overfitting. Nous essaierons de voir si en introduisant une régularisation importante, cet écart pourra être réduit.

ii. Impact du choix de la dimension latente k (dimension commune de U et V)

Nous examinons l'impact de la dimension k sur la qualité de la prédiction de notre méthode.

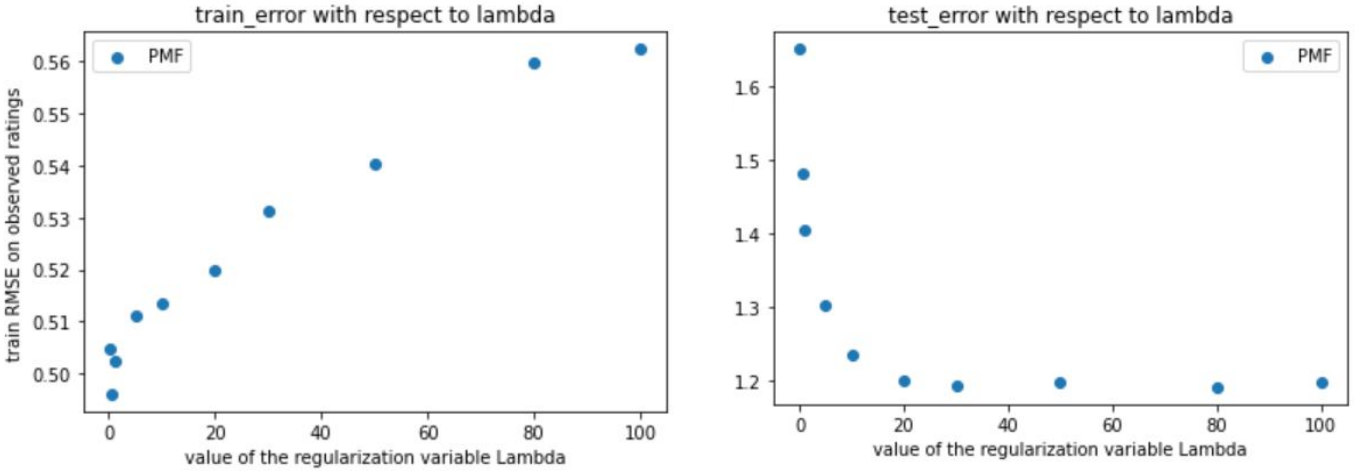




Dans les courbes affichées, nous traçons l'évolution du RMSE selon les valeurs de la dimension latente k . Sur le train, l'erreur diminue quand k augmente, mais sur le test, l'erreur augmente quand k augmente ; une valeur optimale de k est très faible contrairement à la valeur optimale de k dans l'approche ALS qui est au voisinage de 32.

iii. Impact des constantes de régularisation λ_U, λ_V

L'algorithme de PMF a un temps d'exécution très élevé qui est dû à une mise à jour ligne par ligne. Dans cette expérience, nous considérons $\lambda_U, \lambda_V = \lambda$ pour ne pas augmenter l'espace de recherche des hyperparamètres et nous fixons k=10, niter = 20. Nous varions la valeur de λ pour étudier son impact sur le RMSE. Les résultats sont visualisés ci-après :



D'après la figure décrivant l'évolution du RMSE sur le test set, une valeur intermédiaire de λ (λ=10 par exemple) est susceptible de réduire l'effet d'overfitting observé en i .

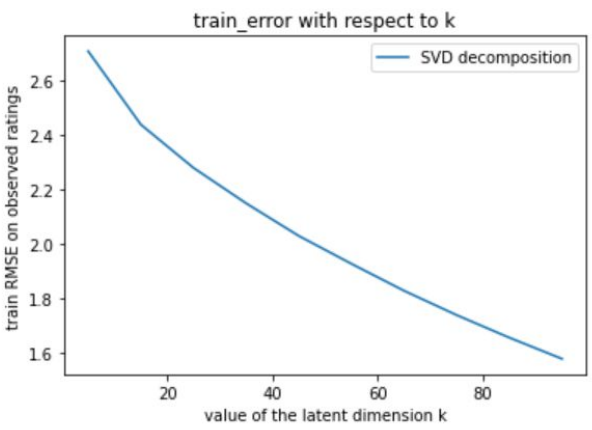
V- Un autre type de factorisation (SVD)

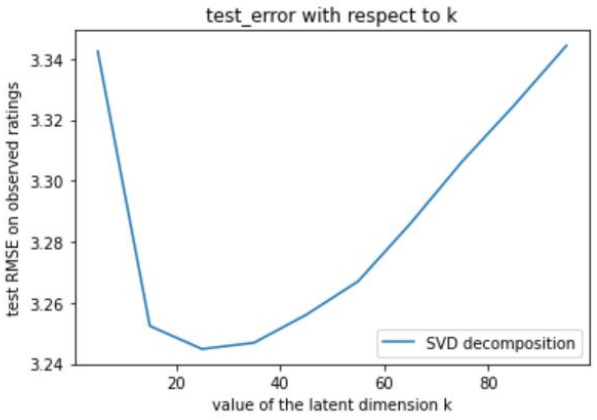
Nous nous sommes penchés vers une formulation un peu différente du problème et qui est comme suit :

Nous cherchons cette fois trois matrices $S, V et U$ avec S matrice diagonale contenant les valeurs singulières de R et de dimension k, U est la matrice des utilisateurs et V est la matrice des films. Ces matrices constituent la "singular value decomposition" de R : $R \approx USV^T$

Un package de Scipy permet directement de retourner ces trois matrices pourvu que nous lui donnions une dimension k.

Nous testons l'impact de cette dimension sur la performance de la prédiction.



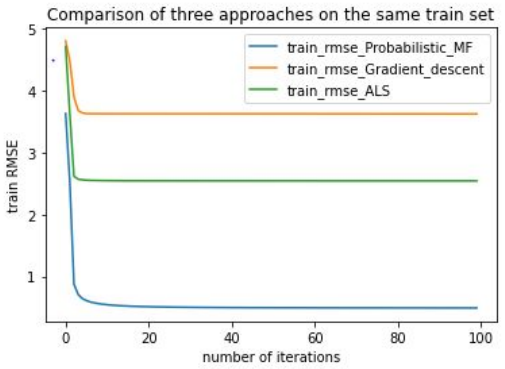


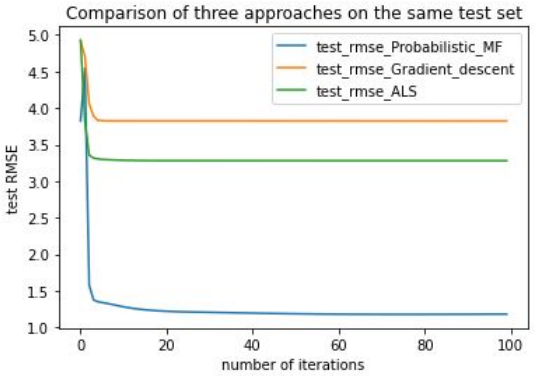
Un comportement similaire au comportement observé pour ALS est mis en évidence. La valeur optimale de k (qui minimise l'erreur sur le test set) est autour de 25 .

VI- Comparaison de différentes approches sur différents datasets :

a- Small dataset

On compare les performances des algorithmes d'apprentissage en premier lieu sur le petit jeu de données MovieLens, comportant 100.000 notes (de 0.5 à 5) de 600 utilisateurs sur 9000 films, comme présentés plus haut. Les résultats sont les suivants, à gauche sur le jeu de données d'entraînement et à droite sur le jeu de données test.

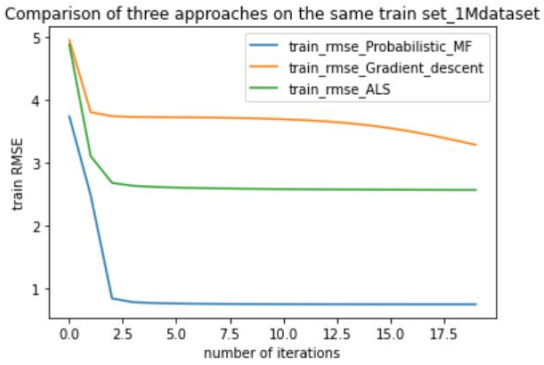


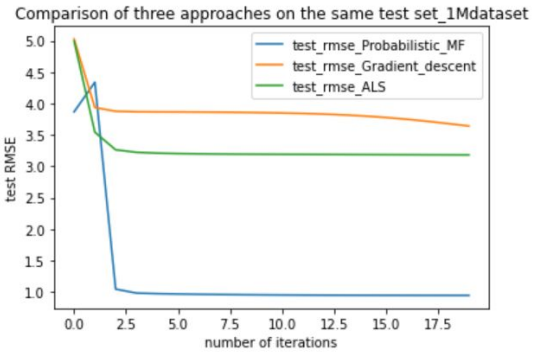


On remarque que la PMF fonctionne bien mieux que la MF classique, que ces matrices sont calculées à l'aide de la descente de gradient ou de l'ALS. On remarque que les trois algorithmes convergent rapidement mais la PMF est plus lente en termes de temps d'exécution que les autres.

b- A slightly larger dataset

La deuxième comparaison s'effectue sur un dataset un peu plus grand (6040 users et 3706 movies). Ce dataset est disponible sur le site https://grouplens.org/datasets/movielens/1m/ . Nous n'avons pas changé les hyperparamètres des algorithmes ni la façon de partager les données en train et test. Nous avons appliqué les trois algorithmes directement au nouveau dataset. Les comportements obtenus sont très similaires à ceux observés pour le small dataset comme le montre les figures suivantes :





Les deux algorithmes ALS et PMF convergent rapidement ; on a l'impression que la descente de gradient n'a pas encore convergé. La PMF a la meilleure performance sur le train et sur le test. La descente de Gradient est moins performante que l'ALS.

Bibliographie :

- Probabilistic matrix factorization : https://towardsdatascience.com/pmf-for-recommender-systems-cbaf20f102f0
- Singular value decomposition : https://hackernoon.com/introduction-to-recommender-system-part-1-collaborative-filtering-singular-value-decomposition-44c9659c5e75
- Alternate least square : http://ethen8181.github.io/machine-learning/recsys/1_ALSWR.html
- MovieLens : https://grouplens.org/datasets/movielens/