ATIGUI Karim

Systèmes de recommandations

I. Introduction

L'objectif de la mise en place d'un système de recommandation est de pouvoir proposer un contenu proche à l'attente de l'utilisateur, dans notre cas, ce sera la recommandation de films.

Il existe deux approches :

- Une approche basée sur des metadats qui caractérisent les utilisateurs et/ou les items, elle s'appelle : *Content Based*
- *Une approche* de filtrage collaboratif qui sert à prédire les actions des utilisateurs en fonction des interactions passés par d'autres utilisateurs, on cherche à rapprocher des utilisateurs par leurs actions pour pouvoir prédire des actions futures.

Pour construire notre système de recommandation ; on va utiliser la deuxième approche, le filtrage collaboratif.

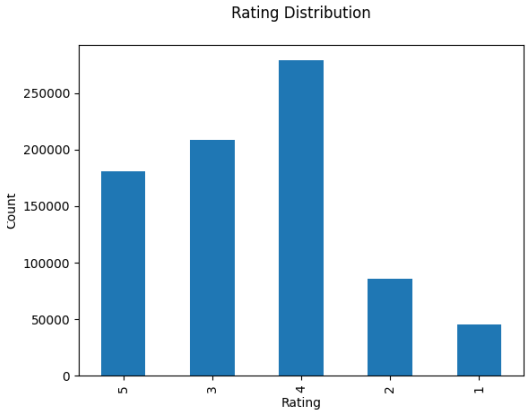
Nous disposons déjà des notes de plusieurs utilisateurs pour plusieurs films, ces notes sont représentées dans une matrice *users/movies*, évidemment le but étant de compléter cette matrice avec les notes 'absentes', lâ ou l'utilisateur n'a pas noté le film, cette note sera prédite par le système de recommandation, si la note est grande, alors le film en question serait proposé à cet utilisateur.

Ce système de recommandation via la complétion de la matrice *users/movies* va donc donner une liste de films à proposer pour chaque utilisateur.

Jeu de donnée

Pour le jeu de donnée, on a 6049 utilisateurs et 3952 films, les notes des utilisateurs vont de 1 à 5. 5 étant la note quand l'utilisateur a apprécié le film.

Le nombre total des notes est 800163, la figure ci-dessous montre la distribution de ces notes.



Afin de compléter la matrice users/items, on utilisera la technique de factorisation matricielle qui consiste à décomposer une matrice en plusieurs matrices.

Factorisation matricielle

Soit ***X*** une matrice (**m** x **n**) qui contient les notes des ***m*** utilisateurs aux ***n*** films.

On décompose la matrice ***X*** en le produit de deux matrices de dimension inférieure, une matrice ***U*** peut être considérée comme la matrice des utilisateurs où les lignes représentent les utilisateurs et les colonnes sont les facteurs latents, l'autre matrice ***V*** est la matrice d'éléments où les lignes représentent les films et les colonnes sont les facteurs latents.

La factorisation matricielle consiste à chercher une approximation de la matrice ***X*** de sorte que :

$$X \approx UV^T$$

- *X* : La matrice des appréciations (ratings)
- *U* : La matrice des préférences des utilisateurs $U \in R^{m \times r}$
- *V* : La matrice des attributs des films $V \in R^{n \times r}$

Le but étant de trouver une représentation ***U*** des utilisateurs et ***V*** des films sous une forme vectorielle, de sorte que la note de l'utilisateur ***i*** donné au film ***j*** soit le produit scalaire de ***Ui*** et ***Vj,*** l'évaluation de l'élément ***i*** donnée par l'utilisateur ***j*** peut être exprimée sous la forme d'un produit scalaire du vecteur latent de l'utilisateur et du vecteur latent de l'élément.

$$\hat{x}_{ij} = \sum_{r=1}^{r} U_{i,r} V_{r,j}$$

L'idée derrière est d'utiliser des facteurs latents pour représenter les préférences de l'utilisateur ou les sujets de films dans un espace de dimension beaucoup plus faible.

Exemple :

Soit la matrice ***X*** suivante qui contient les notes de 6 utilisateurs pour 5 films, on dispose ici de 17 notes :

$$\boldsymbol{X} = \begin{pmatrix} 2 & 3 & 2 & ? & ? \\ ? & 2 & ? & 4 & 3 \\ 3 & ? & 3 & ? & 4 \\ ? & 3 & ? & 4 & 3 \\ ? & ? & ? & ? & 2 \\ 1 & 4 & 3 & 4 & 4 \end{pmatrix}$$

La matrice ***X*** peut être par exemple rapprochée par les matrices ***U*** et $\boldsymbol{V^T}$ suivantes :

$$\mathrm{U} = \begin{pmatrix} 0.7547 & 0.9144 \\ -0.1904 & 1.5834 \\ 1.1320 & 1.3716 \\ 0.5655 & 1.1800 \\ 0.6077 & 0.6636 \\ 1.3214 & 0.7766 \end{pmatrix} \quad \mathrm{V.T} = \begin{pmatrix} -1.0276 & 2.1333 & 1.9117 & 1.4385 & 1.0796 \\ 3.0341 & 1.5196 & 0.6087 & 2.6987 & 2.0242 \end{pmatrix}$$

Avec U (6x2) et V (5x2).

La matrice de prédiction ***Xpred*** rapprochée à la matrice ***X*** sera donc :

$$\text{Xpred} = \begin{pmatrix} 2 & 3 & 2 & 3.55 & 2.66 \\ 5 & 2 & 0.6 & 4 & 3 \\ 3 & 4.5 & 3 & 5.33 & 4 \\ 3 & 3 & 1.8 & 4 & 3 \\ 1.38 & 2.3 & 1.56 & 2.66 & 2 \\ 1 & 4 & 3 & 4 & 3 \end{pmatrix}$$

Ainsi on voit par exemple que le film 1 (colonne 1) peut être proposée à l'user 2 (ligne 2) car la note prédite est 5 et que ce même utilisateur 2 ne va apprécier le film 3.

On voit donc qu'il est possible de construire le système de recommandation en complétant la matrice *users/movies*.

II. Description des modèles utilisés

Pour compléter la matrice *users/movies*, on doit trouver les matrices ***U*** et ***V*** qui nous donnent la matrice de prédiction ***Xpred.***

On va utiliser la fonction objective suivante afin de quantifier la qualité de cette prédiction.

$$\sum_{i,j\in\Omega} (x - x_{Pred})_{i,j}^2 \qquad \Omega \text{ étant l'ensemble des observations}$$

Et en définissant la matrice ***W*** (un masque) afin de ne garder que les observations :

$$\boldsymbol{w_{ij}} = \begin{cases} \mathbf{1} & \boldsymbol{si\ i(i,j) \in \Omega} \\ \mathbf{0} & \boldsymbol{sinon} \end{cases}$$

La fonction objective devient :

$$f(u,v) = \sum_{i=1}^{m} \sum_{j=1}^{n} W_{ij}(X - UV^T)_{ij}^2$$

Notre problème d'optimisation s'écrit finalement.

$$MIN_{\substack{U\in R_{m\,x\,r} \\ V\in R_{n\,x\,r}}} \sum_{i=1}^{m} \sum_{j=1}^{n} W_{ij}(X - UV^T)_{ij}^2 \qquad (1)$$

Régularisation

L'augmentation du nombre de facteurs latents améliore la personnalisation, jusqu'à ce que ce nombre devienne trop élevé, moment auquel le modèle (1) commence à sur-apprendre, on parle de overfitting, pour éviter ce sur-apprentissage, on ajoute des termes de régularisation à la fonction objective.

Le paramètre de régularisation λ est ajouté au modèle (1) qui devient.

$$MIN_{\substack{U\in R_{m\,x\,r} \\ V\in R_{n\,x\,r}}} \sum_{i=1}^{m} \sum_{j=1}^{n} W_{ij}(X - UV^T)_{ij}^2 + \lambda\left(\|U\|_i^2 + \|V\|_j^2\right) \qquad (2)$$

$$MIN_{\substack{U\in R_{m\,x\,r} \\ V\in R_{n\,x\,r}}} \sum_{i=1}^{m} \sum_{j=1}^{n} W_{ij}(X - UV^T)_{ij}^2 + \lambda\left(\sum_{i=1}^{m} U_i^2 + \sum_{j=1}^{n} V_j^2\right) \qquad (2)$$

Contrainte de positivité

Comme les notations sont positifs, il est possible d'ajouter une contrainte de positivité de U et V

U > 0 et V >0

$$MIN_{\substack{U\in R_{m\,x\,r} \\ V\in R_{n\,x\,r} \\ U>0\,,V>0}} \sum_{i=1}^{m}\sum_{j=1}^{n} W_{ij}(X - UV^T)_{ij}^2 + \gamma\left(\sum_{i=1}^{m} U_i^2 + \sum_{j=1}^{n} V_j^2\right) \quad (3)$$

Biais

Les biais introduits par les utilisateurs ou les films pourraient fausser la prédiction, les utilisateurs selon leurs caractères n'attribuent pas des notes de la même façon, des films plus populaires pourraient avoir une note plus généreuse.

Pour contrer ceci, on enrichit le modèle en prenant en compte ces biais, ce qui va normaliser la valeur prédite en tenant compte de ces déviations.

$$b_{ij} = \mu + b_i + b_j$$

- μ : la moyenne des notes données
- b_i : le biais introduit par l'utilisateur i, c'est la différence entre μ et l'ensemble des notes donnée par l'utilisateur i.
- bj : le biais introduit par l'item j, un biais de popularité, la différence entre μ et l'ensemble des notes donnée au film j

La prédiction pour le couple utilisateur i film j n'est donc plus seulement $U_i.V_j$ mais :

$$b_{ji} = \mu + b_i + b_j + U_i.V_j$$

Le modèle avec les biais s'écrit donc :

$$MIN_{\substack{U\in R_{m\,x\,r} \\ V\in R_{n\,x\,r}}} \sum_{i=1}^{m}\sum_{j=1}^{n} W_{ij}(X - \mu - bi - bj - UV^T)_{ij}^2 + \gamma\left(\sum_{i=1}^{m} U_i^2 + \sum_{j=1}^{n} V_j^2 + \sum_{i=1}^{m} b_i^2 + \sum_{j=1}^{n} b_j^2\right) \quad (4)$$

Evaluation

Les matrices U et V sont d'abords initialisées d'une façon aléatoire et ensuite on tente via un algorithme de diminuer progressivement l'erreur entre la valeur réelle et celle de prédiction, le critère utilisé pour l'évaluation de ce modèle sera la racine de l'erreur quadratique moyenne notée *RMSE* : (*Root Mean Square Errror)*

$$\boldsymbol{RMSE}\sqrt{\frac{1}{|\boldsymbol{\Omega}|}\sum_{(\boldsymbol{i},\boldsymbol{j})\in\boldsymbol{\Omega}}\left(\boldsymbol{X_{ij}} - \boldsymbol{U_i V_j}\right)^2} \quad \Omega \text{ étant l'ensemble des observations}$$

III. Description de l'algorithme

Pour résoudre le problème d'optimisation (1), j'ai utilisé deux algorithmes, la descente du gradient et l'algorithme des moindres carrés alternés (ALS) :

La descente du gradient

Pour minimiser la fonction coût, on va calculer sa dérivée pour les paramètres U et V, une mesure de la pente et ensuite on progresse d'un pas $\boldsymbol{\alpha}$ dans la direction de la pente qui descend, soit la direction opposée du gradient. Ces étapes sont répétées en boucle jusqu'à minimisation de la fonction coût (RMSE).

$$\nabla_{f(u_k)} = -2 \cdot \sum_{(i,j)\in\Omega}\left(\boldsymbol{X_{ij}} - \boldsymbol{U_i V_j}\right) \cdot \boldsymbol{V_{jk}} \qquad \nabla_{f(v_k)} = -2 \cdot \sum_{(i,j)\in\Omega}\left(\boldsymbol{X_{ij}} - \boldsymbol{U_i V_j}\right) \cdot \boldsymbol{U_{ik}}$$

Pour la mise à jour des vecteurs U et V :

$$U_{ik\ new} = U - \alpha\, \nabla_{f(u_k)}$$

$$V_{jk\ new} = V - \alpha\, \nabla_{f(v_k)}$$

Algorithme 1 : Descente du gradient

1. *Initialize U, V*
2. *for k=0, …. Do*
3. $U_{k+1} = U_k - \alpha\, \nabla f(U_k)$
4. $V_{k+1} = V_k - \alpha\, \nabla f(V_k)$
5. *end for*

until convergence

Pour un pas fixe, la convergence n'est pas assurée, on utilise donc un pas évolutif (backtracking)

Algorithme 2 : Descente du gradient avec backtracking

1. *Initialize U, V*
2. *Set initial stepsize*
3. *for k = 0, …. Do*
4. $U_{k+1} = U_k - \alpha\, \nabla f(U_k)$
5. *while* $f(U_{k+1}) > f(U_k)$ *do*
6. $\alpha_k = \frac{\alpha_k}{3}$
7. $U_{k+1} = U_k - \alpha\, \nabla f(U_k)$
8. *end while*
9. $\alpha_{k+1} = \alpha_k * 4$
10. *end for*

until convergence

La descente du gradient stochastique

Pour rendre l'algorithme descente du gradient stochastique, on va parcourir le jeu de donnée d'une façon aléatoire et la traverser pour chaque vecteur.

On ne peut plus régler le pas à chaque itération car la direction à chaque itération peut ne pas conduire à une diminution de la fonction de cout globale, on fixe alors un pas petit (0.002).

Algorithme 3 : Descente du gradient Stochastique (SGD)

---

1. *Initialize U, V*
2. *for k = 0, …. do*
3. *shuffle and pick up randomly index* U_k
4. *for n = 0, …. do*
5. $U_{k+1} = U_k - \alpha \nabla f(U_k)$
6. $V_{k+1} = V_k - \alpha \nabla f(V_k)$
7. *end for*
8. *end for*

until convergence

Algorithme des moindres carrés alternés (ALS)

Un autre algorithme pour trouver U et V est l'algorithme des moindres carrés alternés (ALS, alternating least squares) :

Initialiser U et V au hasard, puis fixer U et optimiser V, puis fixer V et optimiser U, etc… jusqu'à une convergence.

Pour mettre à jour V, on calcule le gradient et on l'annule en isolant le terme à mettre à jour

---

Algorithme 4 : Algorithme des moindres carrés alternés (ALS)

---

1. *Initialize U*
2. *for k = 0, …. Do*
3. *update V:* $V = (U^T U)^{-1} \; U^T \; X$
4. *Update U:* $U = (V^T V)^{-1} \; V^T \; X$
5. *end for*

IV. *Présentation, interprétation et discussion des résultats (influence du modèle sur la qualité des prédiction, influence du choix de l'algorithme et de la solution initiale, etc.).*

Appliquons le problème d'optimisation pour la matrice (test) suivante

$$\boldsymbol{X} = \begin{pmatrix} 2 & 3 & 2 & ? & ? \\ ? & 2 & ? & 4 & 3 \\ 3 & ? & 3 & ? & 4 \\ ? & 3 & ? & 4 & 3 \\ ? & ? & ? & ? & 2 \\ 1 & 4 & 3 & 4 & 4 \end{pmatrix}$$

Influence de l'initialisation

On commence par initialiser les matrices U et V avec rank = 1, un seul vecteur type.

- Initialiser U et V d'une façon aléatoire (random)
- Initialiser U et V par des 1 (ones).
- Moyenne des notes (average).

$$U = \begin{pmatrix} 2.3333 \\ 3 \\ 3.3333 \\ 3.3333 \\ 2 \\ 3 \end{pmatrix} \qquad V^T = (1 \quad 1 \quad 1 \quad 1 \quad 1) \qquad \text{Xpred} = \begin{pmatrix} 2.3333 & 2.3333 & 2.3333 & 2.3333 & 2.3333 \\ 3 & 3 & 3 & 3 & 3 \\ 3.3333 & 3.3333 & 3.3333 & 3.3333 & 3.3333 \\ 3.3333 & 3.3333 & 3.3333 & 3.3333 & 3.3333 \\ 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

- SvD.
  On peut utiliser la décomposition en valeurs singulières (SVD) pour obtenir U et V pour les k premiers facteurs de la décomposition correspondant aux k plus grandes valeurs singulières.
  On reconstruit la matrice $\hat{X}$ de la façon suivante :

$$\hat{X} = U\Sigma V^*$$

  - $\Sigma \in R^{k\,x\,k}$ est la matrice diagonale contenant les k premières valeurs singulières de X
  - U $\in R^{m\,x\,k}$ et $V^* \in R^{n\,x\,k}$ sont les matrices contenant les vecteurs singuliers associés aux valeurs singulières de X.

Exemple de décomposition SVD pour la matrice test $\boldsymbol{X} = \begin{pmatrix} 2 & 3 & 2 & ? & ? \\ ? & 2 & ? & 4 & 3 \\ 3 & ? & 3 & ? & 4 \\ ? & 3 & ? & 4 & 3 \\ ? & ? & ? & ? & 2 \\ 1 & 4 & 3 & 4 & 4 \end{pmatrix}$

Pour k=1

$$U = \begin{pmatrix} -0.2599 \\ -0.4747 \\ -0.3208 \\ -0.5281 \\ -0.0854 \\ -0.5640 \end{pmatrix} \quad \Sigma = (10.22) \quad V^* \begin{pmatrix} -0.2002 \\ -0.5449 \\ -0.3105 \\ -0.6132 \\ -0.4366 \end{pmatrix}$$

$$V^T = (\Sigma * V^*)^T = (-2.0460 \quad -5.5691 \quad -3.1739 \quad -6.2670 \quad -4.4624)$$

Pour k =2

$$U = \begin{pmatrix} -0.2599 & 0.2307 \\ -0.4747 & -0.2177 \\ -0.3208 & 0.8743 \\ -0.5281 & -0.2630 \\ -0.0854 & 0.1623 \\ -0.5640 & -0.1987 \end{pmatrix} \quad \Sigma = \begin{pmatrix} 10.22 & \\ & 5.4155 \end{pmatrix} \quad V^* \begin{pmatrix} -0.2002 & 0.5329 \\ -0.5449 & -0.2450 \\ -0.3105 & 0.4595 \\ -0.6132 & -0.5018 \\ -0.4366 & 0.4394 \end{pmatrix}$$

$$V^T = (\Sigma * V^*)^T = \begin{pmatrix} -2.0460 & -5.5691 & -3.1739 & -6.2670 & -4.4624 \\ 2.8856 & -1.3270 & 2.4882 & -2.7175 & 2.3797 \end{pmatrix}$$

La figure ci-dessous représente la RMSE en fonction des initialisations pour un rank-5, à ce stade, il est évident que l'initialisation aléatoire est la plus mauvaise.

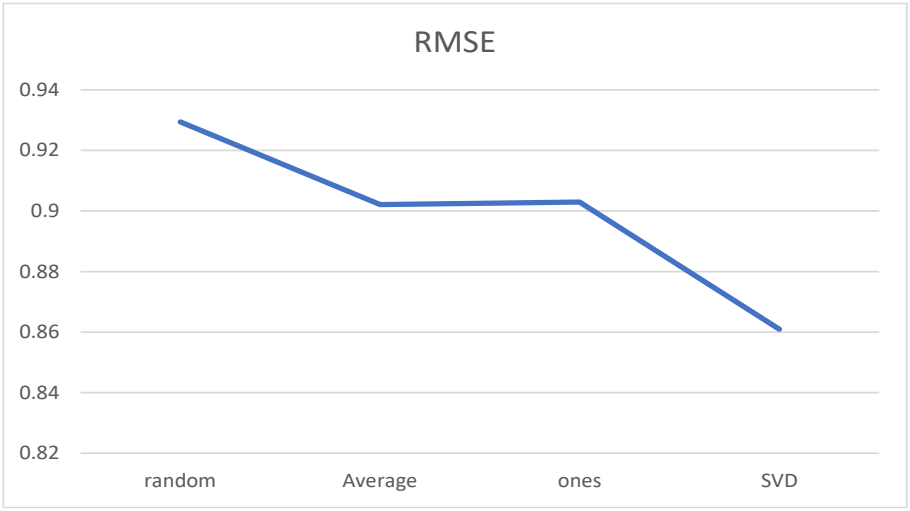
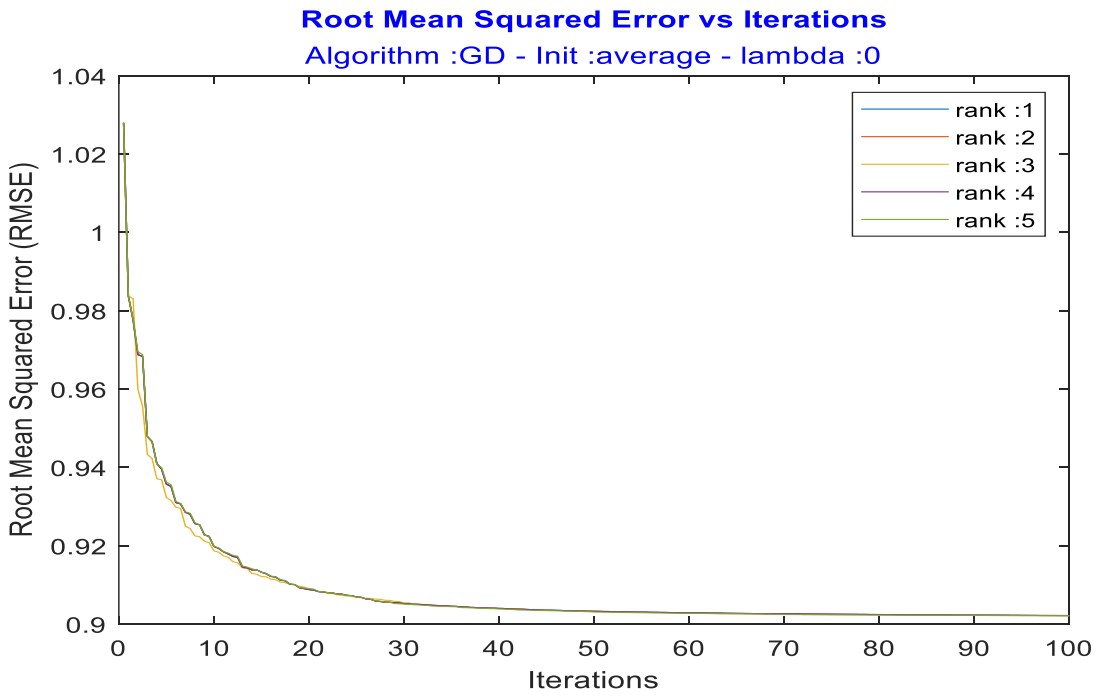


Figure 1 : RMSE (modèle 1) en fonction de l'initialisation (rank-5 et 100 itérations)

Rank

Pour l'initialisation average et ones, le fait d'augmenter les facteurs latents (rank) n'intervient en rien car on a les mêmes vecteurs pour toutes les colonnes.



La figure ci-dessous montre l'évolution du RMSE par rang et par initialisation, on voit que le RMSE s'améliore pour un rang de 5 et que l'initialisation SVD est meilleure pour ce rang.

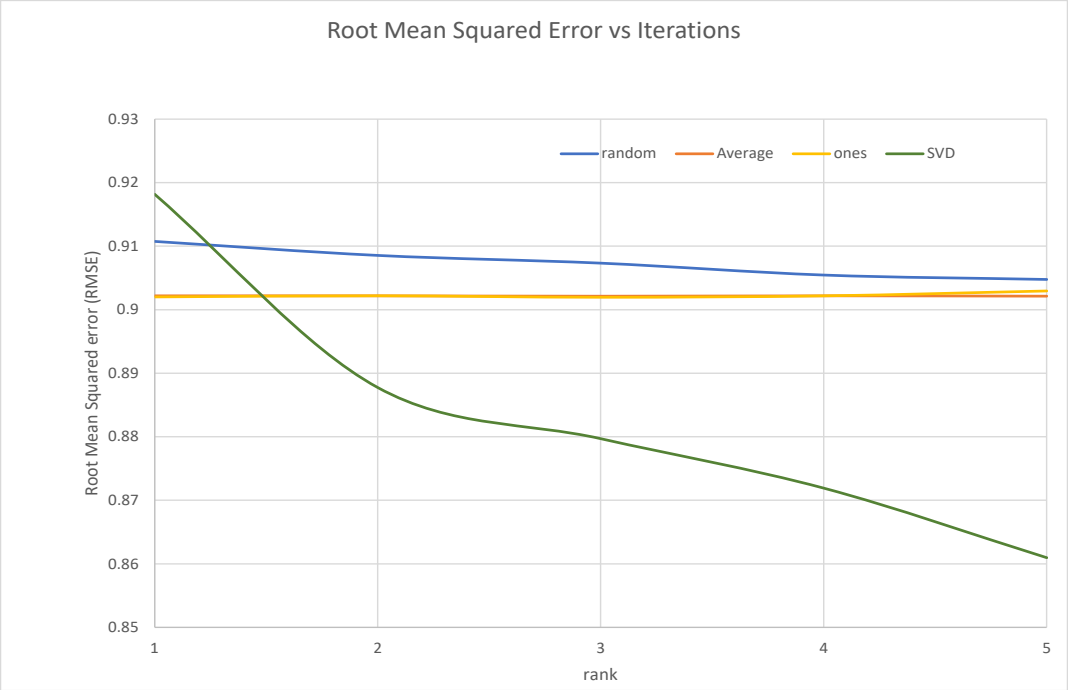


Figure 2 Evolution de RMSE en fonction du rank et de l'initialisation

Pour détecter le problème du surapprentissage et voir à partir de quel rang on commence à rencontrer ce problème, on sépare le jeu de donnée en 2 parties, un jeu de donnée 70% (train) pour l'entrainement et 30% pour la validation (valid), on compare ensuite le RMSE.

On remarque le phénomène d'overfitting à partir du rang 5.

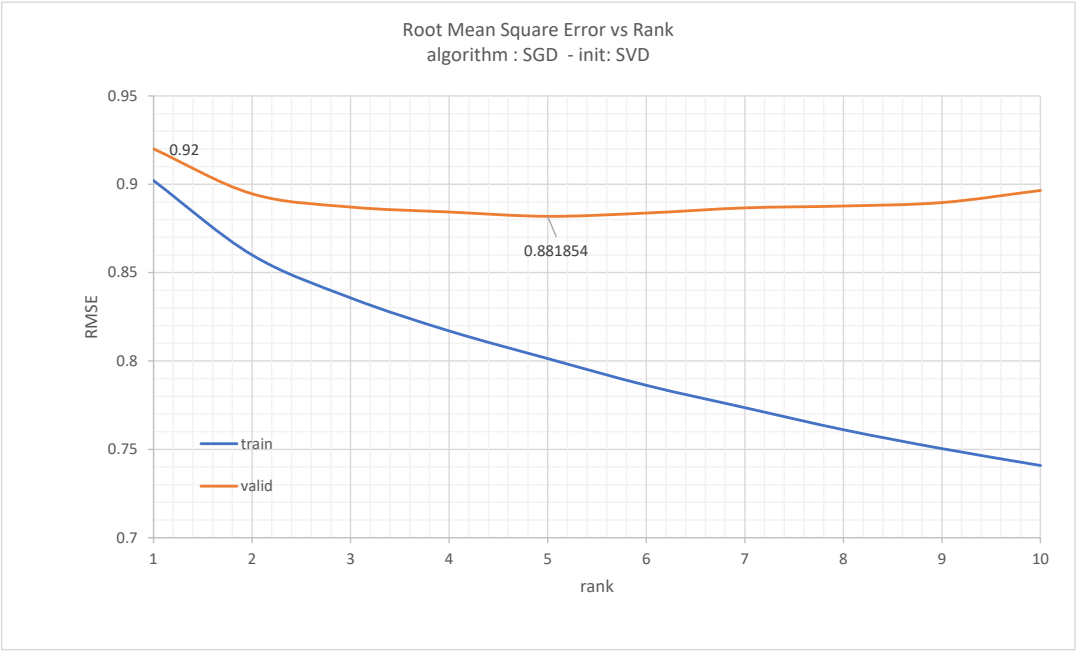


Figure 3 Evolution RMSE par rank pour train et valid data set

Influence du pas d'apprentissage (Learning rate)

Le pas peut être fixé ou évolutif, pour notre jeu de donnée, on voit que le pas doit être plus petit que 0.1 pour ne pas diverger.

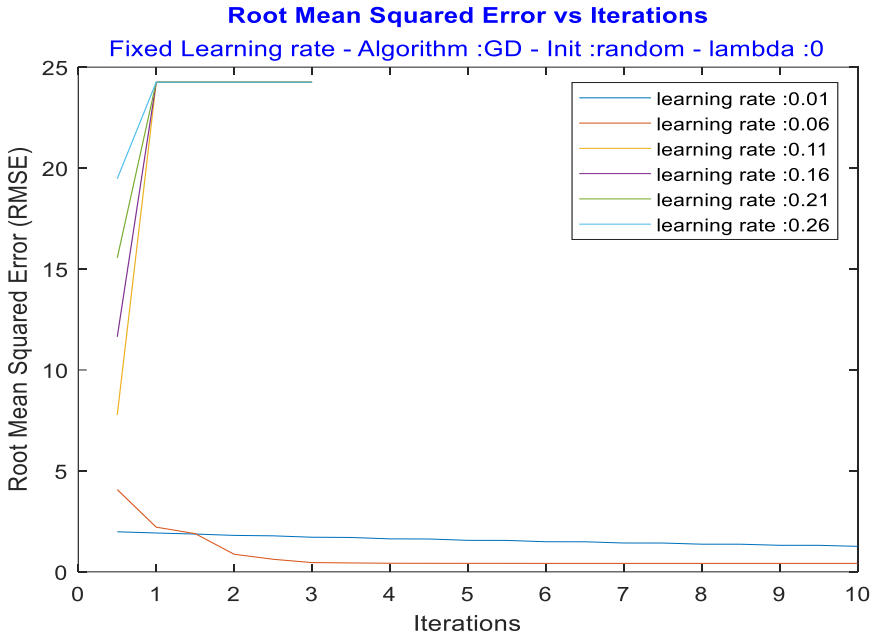
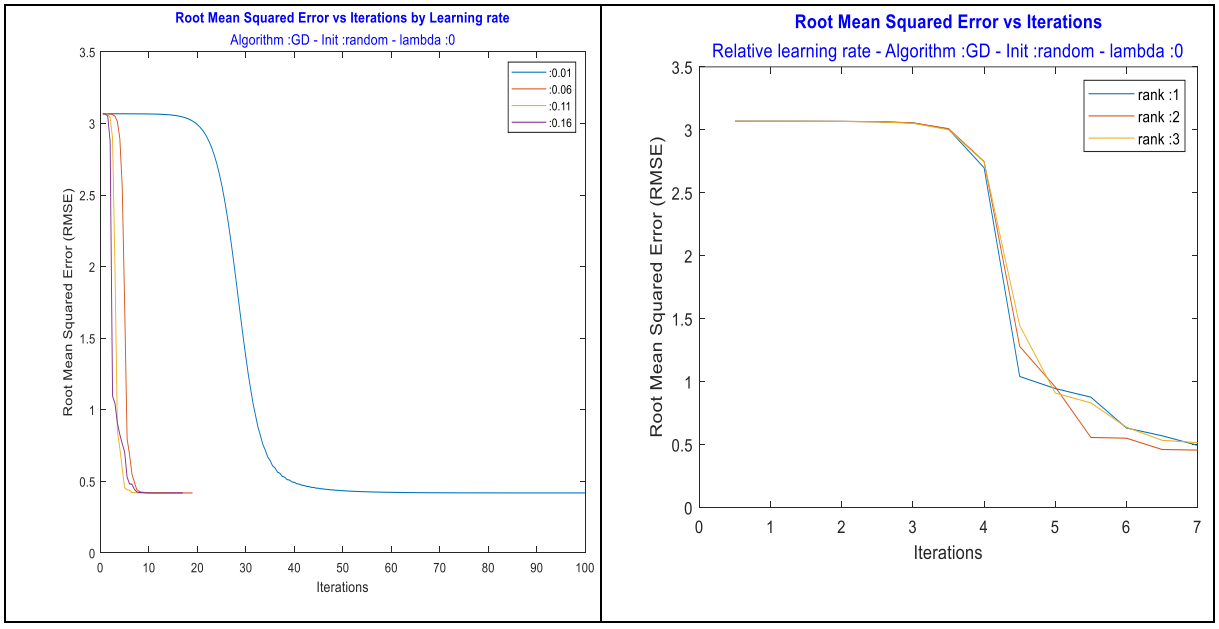


Figure 4Evolution RMSE en fonction du pas d'apprentissage

Pour l'exemple de la matrice test, plus ce pas est petit, plus longue sera la convergence vers un minimum.

Mais si le pas est grand, on risque de passer au-delà d'un minimum local, on part d'un pas relatif à la norme des vecteurs et tant que l'erreur ne diminue pas, on prendra un pas plus petit, la figure de droite montre la convergence avec un pas évolutif.



Pour la descente du gradient, il est judicieux de prendre un pas évolutif qui change à chaque itération (algorithme 2)

Nombre d'itérations et initialisation

La figure ci-dessous montre l'erreur minimum atteinte (RMSE) en fonction du nombre des itérations pour l'algorithme de la descente de gradient, et ce pour une initialisation aléatoire et SVD.

Le tableau ci-dessous montre le nombre maximum atteint des itérations en fonction de l'initialisation et du rang.

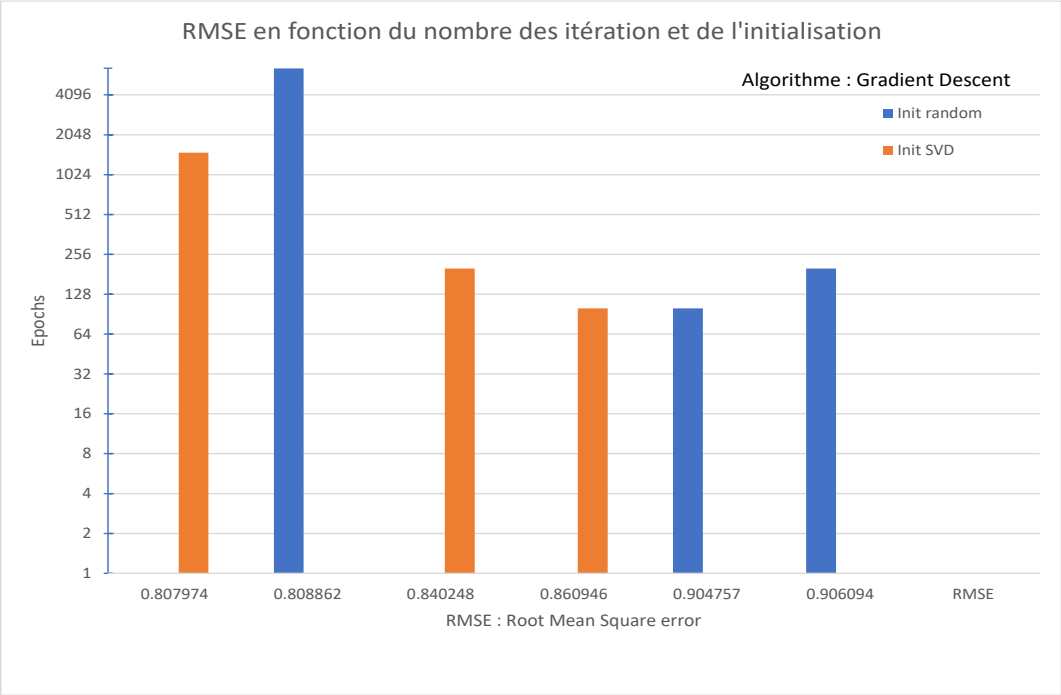


Figure 5 Evolution RMSE (modèle 1) en fonction du nombre d'itérations

L'initialisation SvD est plus efficace, avec un rang 5, on atteint un RMSE correct plus vite que l'initialisation random. (1500 pour SvD vs 6479 pour l'initialisation random)

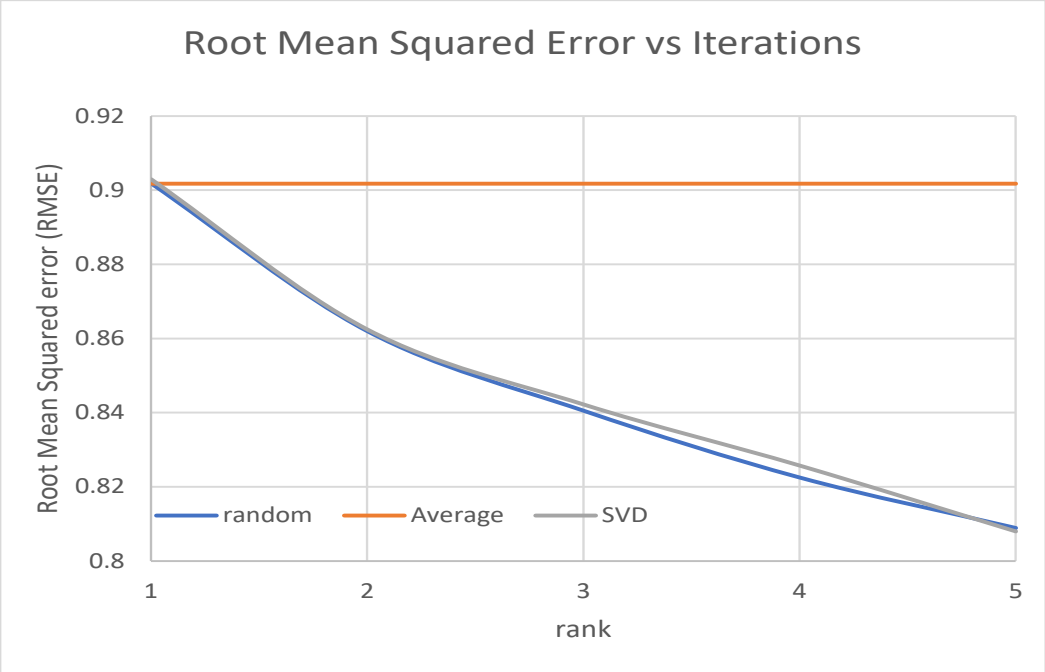
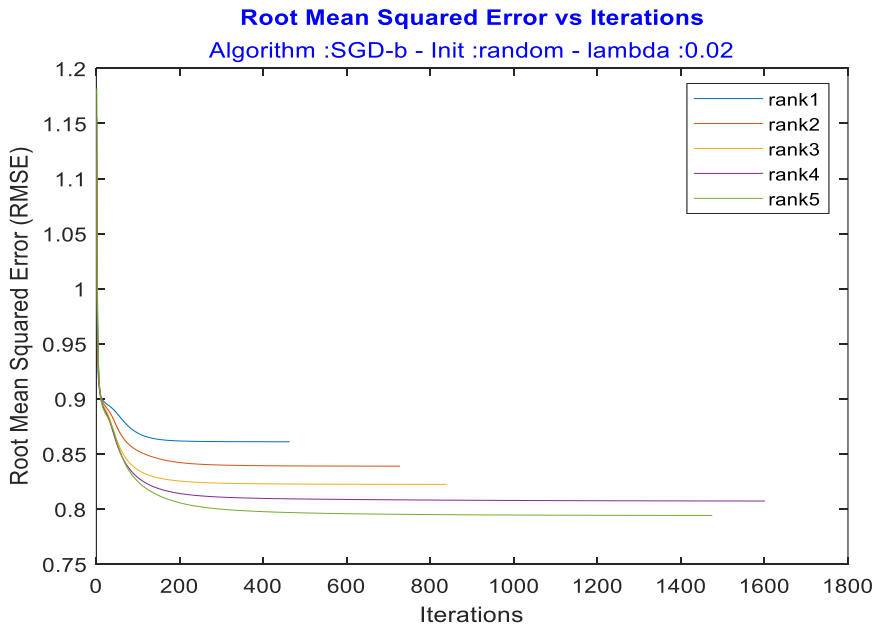


Figure 6 RMSE en fonction du rang jusqu'à convergence (algorithme : gradient descente modèle 1)

La figure ci-dessous montre que le nombre des itérations augmente en fonction du nombre de facteurs latents (rank)



Régularisation

En ajoutant la régularisation au modèle (1), on varie λ et on voit l'influence sur le RMSE pour les jeux de données train et validation.

On déduit de la figure ci-dessous le choix du paramètre λ, qui devait être 0.025 pour éviter le sur-apprentissage.

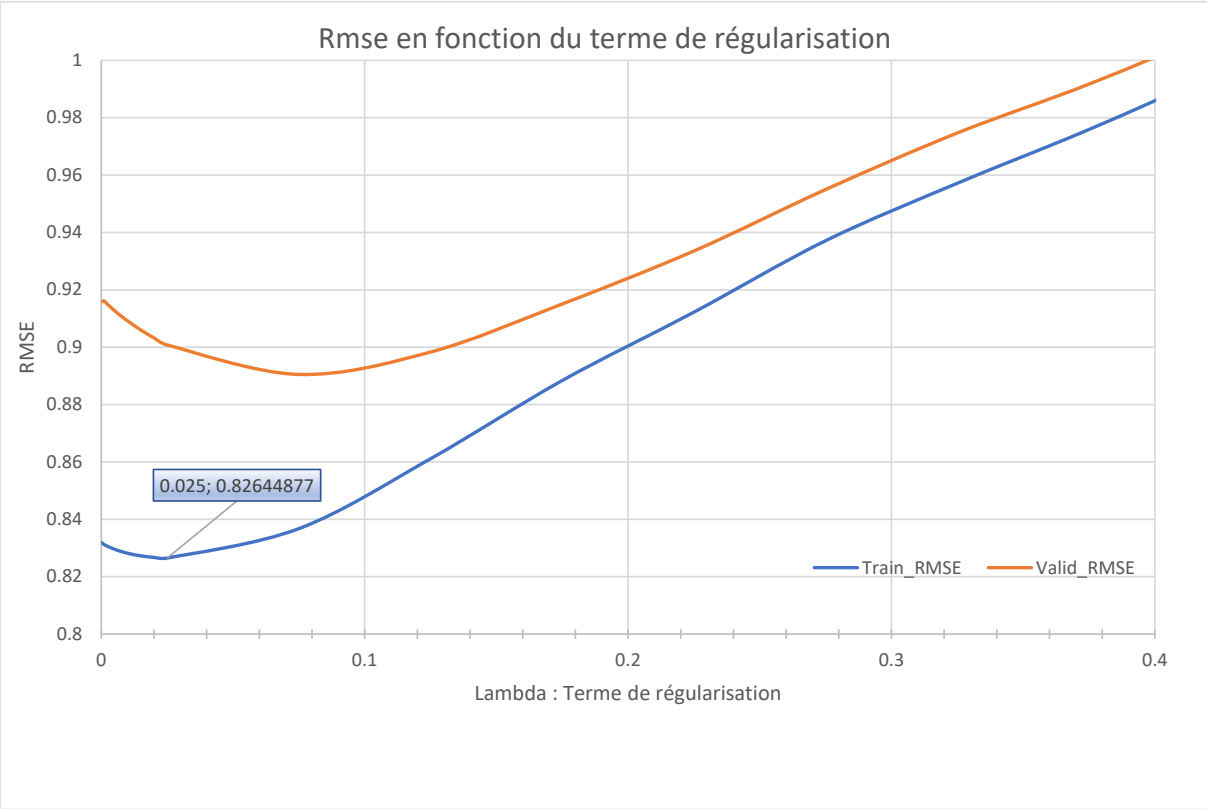
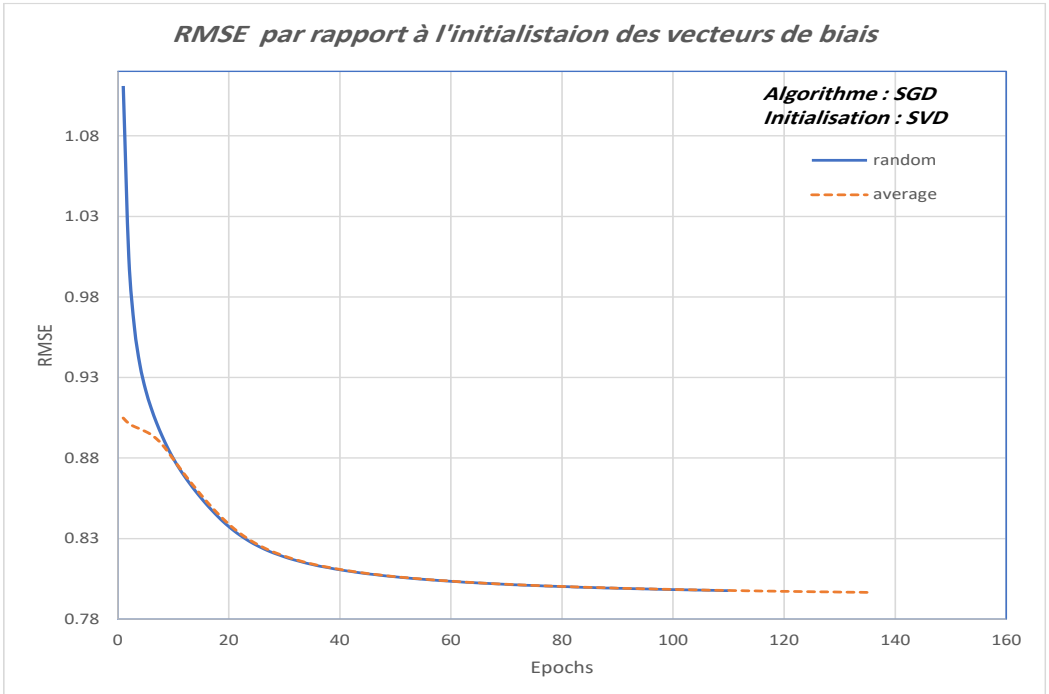


Figure 7: RMSE Train et validation en fonction du terme de régularisation

Biais

Initialisation de vecteurs de biais, soit en aléatoire soit en moyenne, la figurer ci-dessous montre l'influence de cette initialisation sur la RMSE.



Les figures ci-dessous montrent l'évolution RMSE pour les modèles (2) et (4) avec l'algorithme SGD, celle de droite inclut les biais, on remarque l'amélioration par l'introduction de ces biais, on atteint un rmse 0.79 vs 0.81

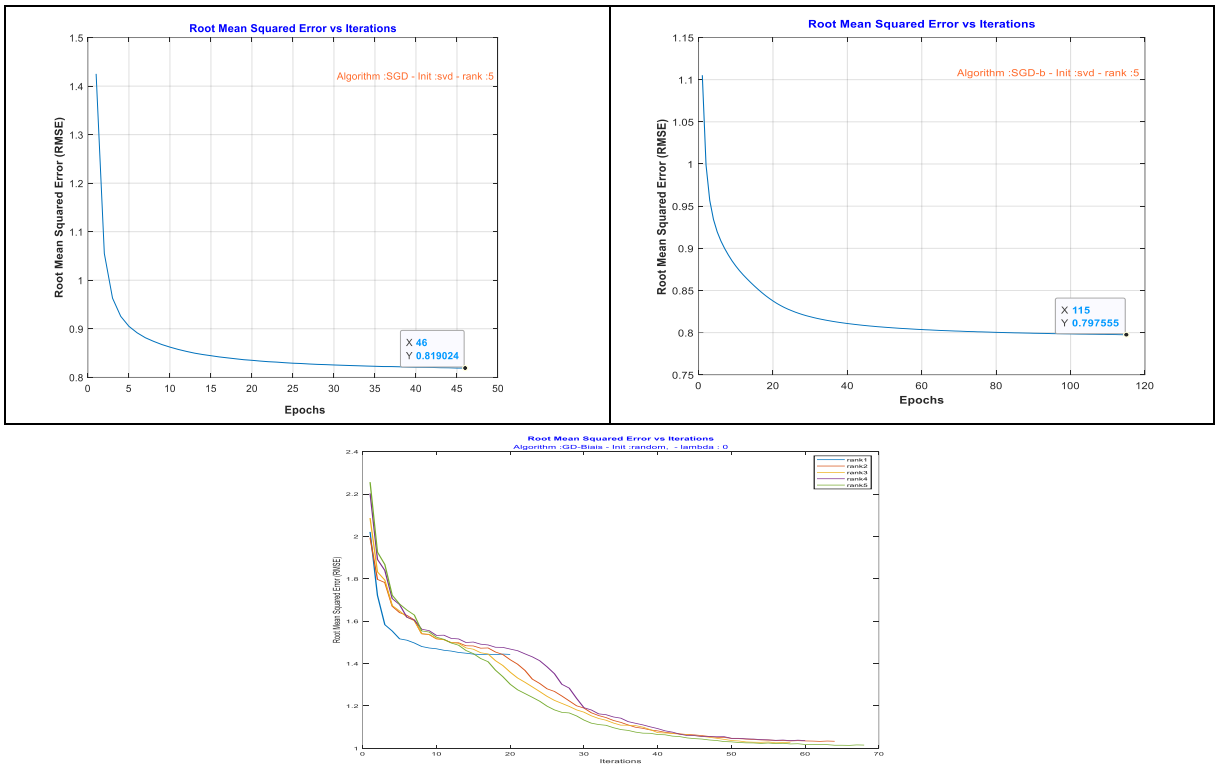


Figure 8 - Evolution RMSE avec l'algorithme descente de gradient pour le modèle 2

Comparaison des algorithmes

La figure ci-dessous montre l'évolution de la RMSE en fonction de l'algorithme de la descente de gradient normal et stochastique, pour les modèles (2) et (4) avec un paramètre de régularisation de 0.02.

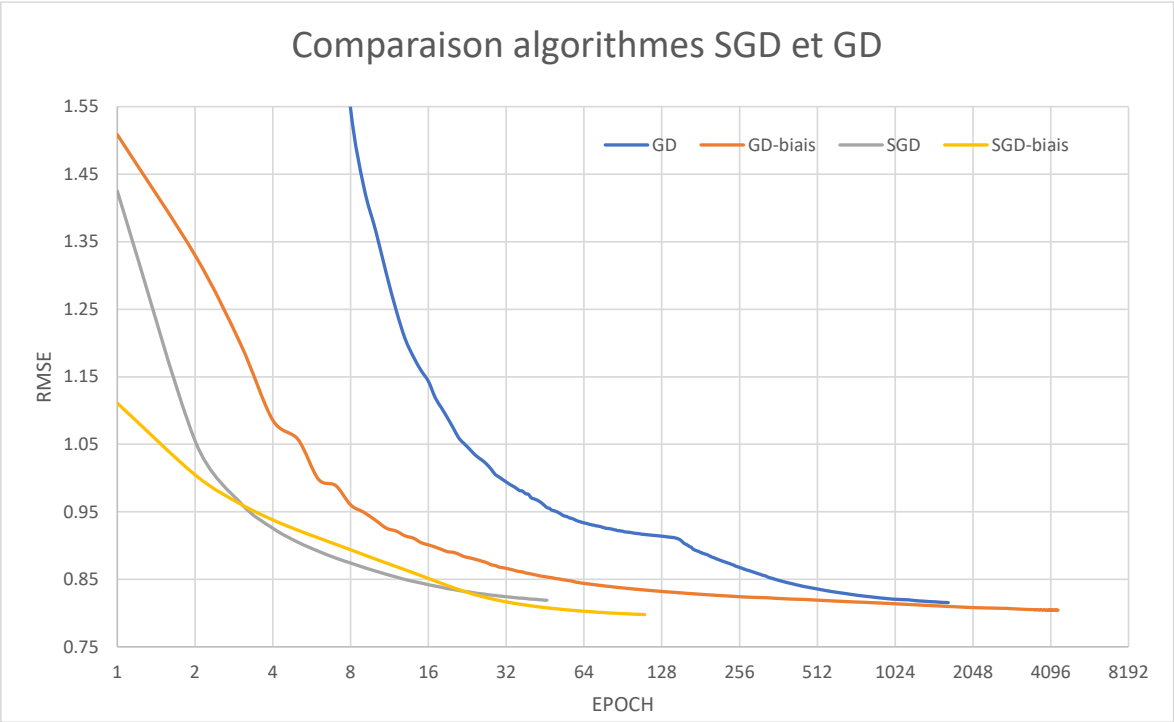


Figure 9 : évolution RMSE en fonction des algorithmes de la descente de gradient (normal et stochastique) avec et sans biais

On voit qu'on obtient un meilleur résultat avec moins d'époch avec l'algorithme : ***SGD avec biais***

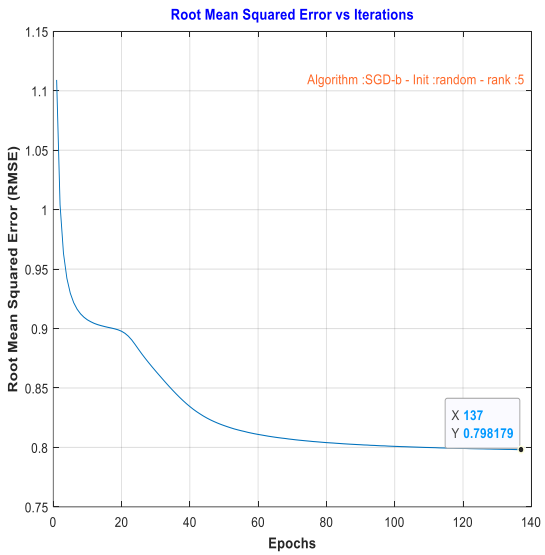


Figure 10 : Evolution RMSE avec SGD et initialisation random

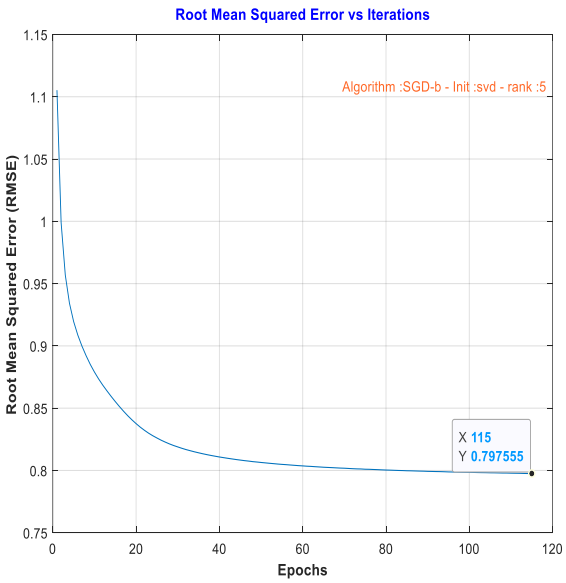
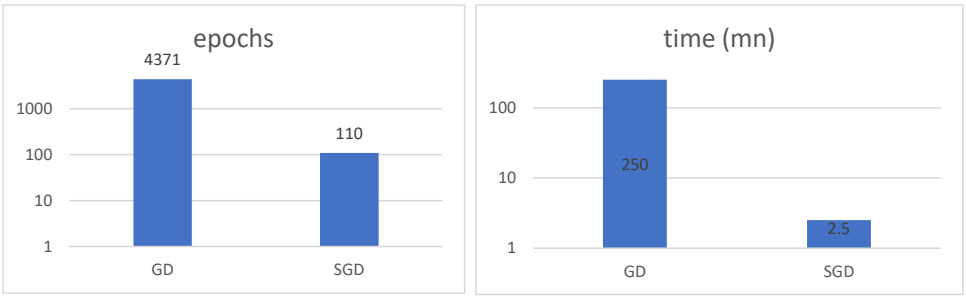


Figure 11 Evolution RMSE avec SGD et initialisation SVD

Temps d'exécution

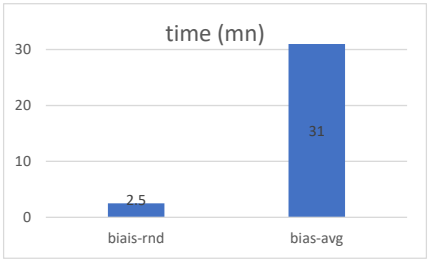
Il y a moyen d'obtenir de bons résultats avec l'algorithme Gradient Descent normal mais celui-ci- est plus lent que l'algorithme SGD, la figure ci-dessous montre le temps d'exécution pour les deux algorithmes.

Algorithm	*epochs*	*time (mn)*	*rmse*	*kaagle*	*init*	*rank*	*λ*
GD	*4371*	*250*	*0.8042*	*0.8704*	*SVD*	*5*	*0.02*
SGD	*110*	*2.5*	*0.7977*	*0.8565*	*SVD*	*5*	*0.02*



On remarque également une différence quant au temps d'exécution en fonction de l'initialisation des biais (random vs average)

Algorithm	*epochs*	*time (mn)*	*train-rmse*	*kaagle*	*init*	*rank*	*λ*	*init biais*
SGD	110	2.5	0.7978	0.8565	SVD	5	0.02	biais-rnd
SGD	137	31	0.7965	0.8576	SVD	5	0.02	bias-avg



Contrainte de non négativité

La figure ci-dessous montre l'influence de la contrainte de non négativité sur le RMSE, modèle (2) vs (3), on ne remarque aucune amélioration pour notre modèle.

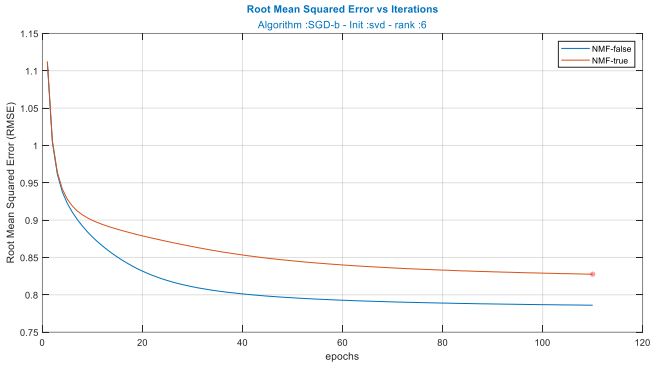


Figure 12: RMSE introduction de la contrainte de Non négativité

V. Projet Matlab

Le système de recommandation a été développé sous Matlab, pour lancer le modèle, il faut exécuter le script *run_me.m* en introduisant les différents paramètres.

```
%% Parameters
 alg = "SGD";       % L'algorithme     : 'GD' - 'SGD' - 'ALS'
 init = svd;  % L'initialisation : 'random' -'ones' -'average' - 'svd'
 biais = true;      % biais : true / false
 lambda = 0.025;    % paramètre de régularisation
 NMF = false;       % NMF : true / false
 lr = 0.002;        % pas d'apprentissage : learning rate
 iter = +inf;       % Nombre d'itérations maximum
 rank = 5;
```

Les fonctions qui interviennent dans l'application sont les suivantes :

- Matric Facto : fonction principale
- Intialise : pour initialiser les matrice U et V
- plotRmse : pour la visualisation graphique
- predict : pour retourner les prédiction (U*V)
- LossFct pour le calcul de l'erreur entre la valeur prédite et la valeur d'origine Xij
- rmse : pour calculer le rmse.
- save_csv : pour générer le ficher csv
- splitTrainValid : pour faire un split dans le jeu de donnée entre un jeu de test et un jeu de validation

Les fonctions des algorithmes :

- GD & GD_biais
- SGD & SGD_biais
- ALS

VI. Conclusion

Pour conclure, voici les paramètres qui ont donné les meilleurs résultats :

- Algorithme : *SGD - biais*
- Rank : *6*
- Initialisation des vecteurs U et V : *SVD*
- Pas d'apprentissage (learning rate) : *0.002*
- Paramètre de régularisation λ : *0.025*
- Initialisation des vecteurs des biais : *aléatoire (random)* car plus rapide