INSTITUT NATIONAL DES SCIENCES APPLIQUÉES DE TOULOUSE

SPÉCIALITÉ MATHÉMATIQUES APPLIQUÉES

Rapport de Projet

Algorithmes de recommandation

Auteures du compte rendu :
EVA ETHEVE
JULIE GONZALEZ
LILA ROIG
Encadrant de projet :
M. FLORENT CHAZEL

Groupe :
MA B
Année Universitaire :
2020/2021

24 mai 2021

TABLE DES MATIÈRES

I Introduction

Depuis quelques années, avec la généralisation d'internet et des nouvelles technologies, la quantité de données disponibles en ligne est devenue extrêmement importante. Ainsi, l'utilisateur peut se retrouver bien souvent submergé par la quantité d'informations car il devient difficile de discerner quelles sont les données pertinentes pour lui et où il peut les trouver. Pour faciliter cette recherche, différents types de systèmes de recommandation ont été mis au point.

Un système de recommandation est une application permettant de filtrer des informations récoltées afin de proposer à un utilisateur des éléments susceptibles de l'intéresser en fonction de son profil.

Aujourd'hui, les systèmes de recommandation sont utilisés dans de nombreux domaines. Par exemple, ils sont très fréquents dans le e-commerce, Amazone utilise notamment un système de recommandation pour proposer à ses clients des produits qu'ils sont susceptibles d'acheter. Cela peut s'avérer très utile pour l'utilisateur, par exemple lorsqu'il est à la recherche d'une nouvelle lecture, mais cela peut également pousser aux achats impulsifs et donc à une surcommation que l'on peut questionner et critiquer. D'autre part, les systèmes de recommandation sont massivement utilisés dans les domaines du streaming vidéo et musical, on peut citer par exemple, l'algorithme de Netflix qui propose des films à ses utilisateurs selon leur profil et selon le contenu, ou encore celui de YouTube. Enfin, ils sont aussi utilisés sur les réseaux sociaux, comme l'algorithme de Facebook, qui, pour chaque utilisateur, classe les publications dans l'ordre qui a le plus de chance de lui plaire.

Il est possible de distinguer principalement quatre différentes approches de conception d'un système de recommandation :

— Approches basées sur la personnalisation
— Approches basées sur le contenu
— Approches basées sur le filtrage collaboratif
— Approches hyrbides

II Les différents systèmes de recommandation

Comme nous l'avons introduit précédemment, il existe donc quatre différentes approches afin de construire des algorithmes de recommandation dont nous décrirons les principales caractéristiques dans cette partie.

II.1 Approches basées sur la personnalisation

La recommandation personnalisée consiste à recommander des objets sur la base du comportement passé de l'utilisateur (historique d'achat et de navigation, clic...). Par exemple, s'il a acheté un produit sur un site de e-commerce, il se verra certainement proposer des produits similaires par des anonces publicitaires.

II.2 Approches basées sur le contenu

L'approche basée sur le contenu consiste à analyser quels sont les éléments du catalogue, qui n'ont pas encore été consultés par l'utilisateur, et qui coïncident le mieux avec ses préférences et intêrets et seraient donc susceptibles de lui plaire. Par exemple, l'algorithme peut recommander à l'utilisateur des éléments qui sont semblables à ce qu'il a déjà acheté avant. Il utilise pour cela des indicateurs de similarité. Par exemple, si notre algorithme de recommandation porte sur des livres, on peut caractériser chaque livre par une liste de mots clés (on se limite en général à une centaine de mots clés différents), et l'algorithme calcule leur fréquence d'utilisation dans le livre. Chaque mot est associé à un poids qui caractérise son importance et ces données sont ensuite croisées avec la liste des mots clés préférés de l'utilisateur pour déterminer le coefficient de Dice (qui est un indicateur de similarité). Les approches basées sur le contenu peuvent aussi utiliser des systèmes de notation qui leur permettent d'évaluer la satisfaction du client vis-à-vis d'un article. Par exemple, la méthode de retour de pertinence de Rocchio permet de classer les notations de l'utilisateur en deux groupes (notation positive et notation négative) afin d'affiner sa recherche.

Le principal avantage de l'approche par contenu est qu'elle ne nécessite pas une large communauté d'utilisateurs puisqu'on ne les compare pas entre eux. De plus, il n'y a pas de problème de démarrage à froid à l'ajout d'un nouveau produit car il suffit de dresser la liste de ses caractéristiques pour pouvoir ensuite les faire coïncider avec les préférences de l'utilisateur. Cependant, il est parfois difficile d'extraire les caractéristiques d'un produit ce qui peut entraîner un coût supplémentaire pour construire la base de données. Par ailleurs, ce type d'approche nécessite que le profil de l'utilisateur soit assez renseigné avec la liste de ses préférences, il faut donc un

historique du système suffisamment conséquent. Pour éviter un démarrage à froid à l'arrivée d'un nouvel utilisateur, certains algorithmes demandent aux utilisateurs de rentrer une première indication de leur préférence. Par exemple, l'algorithme de Pandora (sur des recommandations musicales) demande en premier lieu à l'utilisateur d'indiquer son artiste préféré. Il peut ainsi orienter la liste de ses propositions vers des morceaux du même artiste ou de style similaire.

II.3 Approches basées sur le filtrage collaboratif

La technique du filtrage collaboratif consiste à réaliser des recommandations en se basant sur le comportement passé d'utilisateurs ayant des préférences et intérêts similaires. C'est notamment cette approche que nous tenterons de reproduire dans ce projet. Elle nécessite principalement une base de données suffisamment fournie quant aux préférences des utilisateurs. On distingue notamment deux types d'approche par filtrage collaboratif : l'approche *Item-to-Item* et l'approche *User-to-User*. Premièrement, l'approche *Item-to-Item* (aussi appelée *model-based*) est basée sur le degré de similarité entre les éléments, l'algorithme recherche d'abord des contenus similaires puis les propose à l'utilisateur. D'autre part, l'approche *User-to-User* (aussi appelée *memory-based*) est basée sur le degré de similarité entre les utilisateurs. A partir des données, l'algorithme dégage un sous-groupe dont les préférences sont similaires et pour lesquelles il va calculer une moyenne pondérée sur l'ensemble du sous-groupe. Chaque individu est associé à un certain poids, qui dépend de son degré de corrélation avec l'utilisateur considéré. A partir de la fonction de préférence qui en résulte, l'algorithme va faire de nouvelles recommandations à l'utilisateur. Par exemple, si dans le groupe plusieurs utilisateurs ont aimé le même livre, alors l'algorithme recommandera également ce livre aux individus du même groupe ne l'ayant pas encore acheté.

Un des avantages du filtrage collaboratif est que plus il y a d'utilisateurs, plus il y a de notations des produits et donc meilleures sont les prédictions. Cependant, on observe souvent que parmi les utilisateurs qui achètent un produit, peu nombreux sont ceux qui vont se donner la peine de laisser une note. D'autre part, ce système présente le problème du démarrage à froid à l'arrivée d'un nouvel utilisateur car ses préférences sont inconnues, mais aussi à l'ajout d'un nouveau produit dans le catalogue puisque personne ne lui a encore attribué de note. Enfin, un autre incovénient du système est qu'il va avoir tendance à proposer toujours le même type d'éléments à un utilisateur donné sans chercher à diversifier ses propositions.

Cette méthode est utilisée notamment par les géants Facebook et Amazon. L'algorithme d'Amazon est basé sur l'approche *Item-to-Item*. Il met en relation chacun des produits que l'utilisateur a acheté et noté avec des produits similaires dont il va pouvoir faire une liste de recommandation.

II.4 Approches hybrides

L'approche hybride consiste à combiner les deux approches précédentes de filtrage par contenu et filtrage collaboratif afin d'en tirer tous les avantages et d'essayer d'en atténuer les inconvénients. Il existe plusieurs types de combinaisons différentes. Premièrement, la combinaison monolythique consiste à intégrer les aspects des différents systèmes de recommandation en un unique algorithme. D'autre part, la combinaison parallèle consiste à utiliser séparemment plusieurs types d'algorithmes qui vont produire chacun une liste de recommandation. Celles-ci seront ensuite recoupées pour fournir une unique liste en sortie. Enfin, la combinaison tubulaire consiste à mettre les algorithmes en cascade, la sortie d'un algorithme devient l'entrée du suivant.

Par exemple, l'algorithme d'Amazon est un système hybride. Les recommandations se basent à la fois sur l'historique de naviguation et d'achat de l'utilisateur, mais aussi sur le contenu de l'article qu'elles comparent avec d'autres articles similaires, et avec des articles achetés par des individus similaires.

III Implémentation d'algorithmes basés sur le filtrage collaboratif pour une recommandation d'items

III.1 Objectif du projet de modélisation

On se propose ici de réaliser des algorithmes basés sur la technique du filtrage collaboratif afin de trouver les meilleurs scores de prédiction possibles pour des notations de films. Ce projet s'inspire d'un concours organisé par Netflix en 2011. Nous disposons ainsi de deux jeux de données. Le premier est tiré de la plateforme de recommandation cinématographie *Movielens*, et le deuxième de la plateforme *Jester jokes*. Pour l'implémentation de ces différents algorithmes, nous utiliserons le langage de programmation *Python*.

III.2 Description du jeu de données

Nous testerons nos programmes sur plusieurs jeux de données présentés ci-dessous.

Le jeu de données MovieLens

Le jeu de données Movielens regroupe exactement 100 836 notations allant de 0.5 (film très peu apprécié) à 5 étoiles (film très apprécié). Il comporte 610 utilisateurs et 9742 films. Il est constitué de différents fichiers :

- *ratings.csv* : contient les 100 836 notes (*rating*) attribuées aux films (*movieId*) par les 610 utilisateurs (*userId*). La dernière donnée de ce fichier est (*timestamp*) qui correspond aux secondes après minuit.
- *movies.csv* : contient la liste des 9742 films avec leurs caractéristiques (l'identifiant *movieId*, le titre *title*, le genre *genres*)
- *links.csv* : contient tous les identifiants de tous les films ce qui permet de les identifier sur d'autres plateformes de streaming. Ici, chaque ligne contient les informations suivantes : *movieId* utilisé par https.//movielens.org, *imdbId* utilisé par https.//www.imdb.com, *tmdbId* utilisé par https.//www.themoviedb.org.
- *tags.csv* : ce dernier fichier contient les tags donnés par les utilisateurs aux films i.e des mots clés qui décrivent le film qu'ils ont vu (Exemple : *funny*, *Leonardo Dicaprio*...)

Nous pouvons d'ailleurs faire quelques remarques importantes pour la suite lors de l'implémentation de nos programmes :

- Chaque utilisateur a noté au moins 20 films.
- Les films ne sont pas systématiquement notés i.e il en existe sans notation.
- Les identifiants des films ne vont pas de 1 à 9742, en effet nous remarquons que certains

films ont des identifiants aux alentours de 193 000 par exemple.

Les jeux de données Jester Dataset

Les jeux de données Jester Dataset se présentent sous la forme d'un fichier excel contenant une matrice représentant les notations de blagues par des utilisateurs. Les numéros des lignes correspondent aux identifiants des utilisateurs tandis que les numéros des colonnes correspondent aux identifiants des blagues. La première colonne de la matrice contient le nombre de blagues évaluées par chaque utilisateur. Enfin, chaque note est un nombre réel compris entre -10 et 10. Le nombre 99 correspond à une absence de notation.

— Le jeu de données Jester Dataset4 comporte 105948 notations de 158 blagues par 7699 utilisateurs. 22 blagues n'ont pas d'évaluation et leurs identifiants sont : 1, 2, 3, 4, 5, 6, 9, 10, 11, 12, 14, 20, 27, 31, 43, 51, 52, 61, 73, 80, 100, 116. De plus, 10 utilisateurs n'ont noté aucune blague.

— Le jeu de données Jester Dataset11 comporte 1810455 notations de 101 blagues par 24983 utilisateurs. Tous les utilisateurs ont noté 36 blagues ou plus.

— Le jeu de données Jester Dataset12 comporte 1708993 notations de 101 blagues par 23500 utilisateurs. Tous les utilisateurs ont également noté 36 blagues ou plus.

III.3 Modélisation du problème

On commence d'abord par modéliser notre problème, base qui servira pour toutes les parties qui vont suivre. On introduit alors la matrice utilisateur-item :

$$R = \begin{pmatrix} r_{1,1} & \cdots & r_{1,n} \\ \cdots & r_{u,i} & \cdots \\ r_{m,1} & \cdots & r_{m,n} \end{pmatrix}$$

où le coefficient $r_{u,i}$ représente la note attribuée par l'utilisateur u pour l'item i. S'il n'existe pas de note pour l'item i le coefficient vaudra 0 ou 99 selon le jeu de données considéré. Les entiers n et m sont respectivement le nombre d'items et le nombre d'utilisateurs. Etant donné que sur *Python*, les indices des matrices commencent à 0 et sont des entiers naturels consécutifs, pour le jeu de données Movielens, nous avons donc créé un dictionnaire permettant de faire le lien entre l'identifiant des films/utilsateurs avec leur indice dans la matrice.

On remarque d'ailleurs que cette matrice est très creuse car en pratique les utilisateurs notent très peu d'items. Pour Movielens par exemple, nous ne disposons que de 100 836 notations soit environ 1,7% de la capacité maximale de notre matrice.

III.4 Evaluation de la performance des algorithmes

Pour évaluer la performance de nos algorithmes de prédiction, nous nous baserons sur deux métriques, la MAE (*Mean Absolute Error - Erreur Absolue Moyenne*) et la RMSE (*Root Mean Squared Error - Racine de l'Erreur Quadratique Moyenne*).

$$MAE(Test) = \frac{\sum_{u,i,r_{ui} \in Test} |S(u,i) - r_{ui}|}{|Test|} \tag{1}$$

$$RMSE(Test) = \sqrt{\frac{\sum_{u,i,r_{ui} \in Test} (S(u,i) - r_{ui})^2}{|Test|}} \tag{2}$$

où *Test* est un ensemble contenant les notes réelles r_{ui} données par les utilisateurs u aux items i.

Ces deux indicateurs mesurent la taille de l'erreur commise en comparant la note prédite par l'algorithme à la note réelle donnée par l'utilisateur. Plus le score est faible, meilleure est la prédiction. Pour les appliquer à un jeu de données, il faut au préalable supprimer 20% des notes contenues dans la matrice R et les stocker ailleurs en mémoire. Nous obtenons ainsi la matrice R_{train} sur laquelle l'algorithme est entraîné pour prédire les notes qui ont été supprimées.

Le score RMSE pénalise davantage les grandes erreurs de prédiction, c'est-à-dire que si pour une note l'algorithme fait une très mauvaise prédiction, celle-ci impactera beaucoup plus le score RMSE que le score MAE. Le score RMSE est donc généralement préféré au score MAE pour évaluer et comparer les performances de plusieurs algorithmes.

IV Prédicteur de base

Les prédicteurs de base (*baseline predictors*) sont les algorithmes de recommandation les plus faciles à implémenter, mais également les moins précis. Ils indiquent un seuil de performance minimum que tous les nouveaux algorithmes de recommandation doivent dépasser pour pouvoir être considérés comme valables. Malgré le fait qu'ils fassent partie des algorithmes les moins performants, ils sont utilisés comme algorithmes de secours quand un algorithme plus avancé échoue à établir une prédiction sous des conditions extrêmes comme l'arrivée de nouveaux utilisateurs ou de nouveaux items dans la base de données. Par la suite nous verrons par exemple l'utilisation que nous en avons fait dans l'algorithme des plus proches voisins.

IV.1 Formules mathématiques et implémentation de l'algorithme

Dans cette partie, nous expliquerons les formules mathématiques utilisées par les prédicteurs de base pour calculer la prédiction et comment nous les avons implémentées.
La prédiction consiste à calculer pour un utilisateur u et un item donné i, la note supposée que celui-ci lui aurait accordé. Le score se calcule de la manière suivante :

$$S(u,i) = \mu + b_u + b_i \tag{3}$$

où μ est la moyenne de l'ensemble des notes de notre base de données, b_u est le biais de l'utilisateur u et b_i est le biais de l'item i.

La moyenne μ se calcule en sommant les notes contenues dans la matrice R, puis en divisant par le nombre total de notes :

$$\mu = \frac{\sum_{r_{ui} \in R} r_{ui}}{|R|} \tag{4}$$

Le biais b_i indique si l'item a été plus ou moins bien noté que la moyenne générale de l'ensemble des items. Pour le calculer, on parcourt le set U_i des utilisateurs ayant noté l'item, on fait la somme de la différence entre la note accordée et la moyenne, puis on pondère par le nombre de notes reçues par l'item :

$$b_i = \frac{\sum_{u \in U_i} (r_{ui} - \mu)}{|U_i|} \tag{5}$$

Le biais de l'utilisateur b_u indique si l'utilisateur a tendance à donner des notes plus ou moins élevées que la somme de la moyenne et du biais des items. Pour le calculer, on parcourt le set I_u des items notés par l'utilisateur et on fait la somme de la différence entre la note accordée, la

moyenne et le biais, puis on pondère par le nombre total de notes données par l'utilisateur :

$$b_u = \frac{\sum_{i \in I_u}(r_{ui} - b_i - \mu)}{|I_u|} \tag{6}$$

Un utilisateur n'ayant donné aucune note, ou un item n'ayant jamais été noté ont par défaut un biais nul.

Enfin, l'introduction d'un terme d'amortissement β permet de contrebalancer les cas où il y a peu de notes pour un utilisateur ou un film. Ainsi, les biais se calculent de la manière suivante.

$$b_i = \frac{\sum_{u \in U_i}(r_{ui} - \mu)}{|U_i| + \beta} \text{ et } b_u = \frac{\sum_{i \in I_u}(r_{ui} - b_i - \mu)}{|I_u| + \beta} \tag{7}$$

Notre implémentation du prédicteur de base se découpe ainsi en trois fonctions. La première fonction permet de précalculer les biais de tous les utilisateurs et de tous les items (Cf. équations 7 et 6) et de les stocker dans deux listes, afin d'optimiser notre algorithme. La deuxième fonction permet le calcul du score (Cf. équation 3), elle inclut cependant une correction. En effet, il arrive que le score calculé soit légèrement supérieur à 5 ou inférieur à 0.5. Dans ces cas là on force le résultat à la limite car le jeu de données *Movielens* ne contient que des notes entre 0.5 et 5. La troisième teste la fonction de score sur les 20% de données manquantes et compare la prédiction aux valeurs réelles.

IV.2 Performance du prédicteur de base

La troisième fonction précedemment mentionnée calcule les scores MAE et RMSE obtenus sur le jeu de données. Sur le jeu de données Movielens, lorsque l'on teste l'algorithme sur les 20% de données supprimées, le score MAE obtenu est de 0.6889 et le score RMSE est de 0.8995. Avec l'introduction d'un terme de damping β fixé à 20, les scores diminuent avec un MAE de 0.6833, et un RMSE de 0.8836.

Le tableau ci-dessous récapitule les résultats obtenus sur les différents jeu de données .

Jeu de données	MAE	RMSE
MovieLens	0.6648	0.8814
Jester Dataset4	3.4834	4.5683
Jester Dataset11	3.5355	4.3788
Jester Dataset12	3.5568	4.4053

En conclusion, l'avantage principal du prédicteur de base est qu'il est un algorithme robuste et facile à implémenter. En revanche, la prédiction qu'il fournit est relativement faible, elle sert de seuil de référence pour déterminer si un type d'algorithme est adapté pour faire une prédiction sur le jeu de données.

V Algorithme des plus proches voisins

L'algorithme des plus proches voisins (*nearest neighbors algorithm*) fait partie des premiers algorithmes de filtrage collaboratif à avoir été implémentés. Il existe deux types d'approches pour cet algorithme. D'une part, l'approche *user-user* (utilisateur-utilisateur) consiste à trouver les utilisateurs similaires à l'utilisateur u pour lequel on veut faire une prédiction. Les utilisateurs voisins sont des utilisateurs qui ont acheté par le passé les mêmes items que l'utilisateur u et qui leur ont donné une note équivalente. D'autre part, l'approche *item-item* consiste à trouver des items voisins de celui pour lequel on veut faire une prédiction. Dans les deux approches, à partir du set de plus proches voisins, on collecte les notes attribuées par les utilisateurs aux items et on les utilise pour prédire la note qu'aurait donné l'utilisateur u à l'item i considéré. Ces algorithmes sont une technique standard du machine learning.
Dans cette partie, nous présenterons notre implémentation de l'approche *user-user*. Ce type d'algorithme de recommandation était assez populaire dans les premiers systèmes de recommandation, mais il a perdu de sa popularité en raison des problèmes d'extensibilité dans les systèmes avec de nombreux utilisateurs.

V.1 Formules mathématiques et implémentation de l'algorithme

Calcul des similarités entre utilisateurs

Un des premiers objectifs de l'algorithme est d'établir le set N_u des plus proches voisins de l'utilisateur u. Pour cela, il faut calculer la similarité de l'utilisateur u à tous les autres utilisateurs v du jeu de données. La similarité se calcule de la manière suivante :

$$sim(u,v) = \frac{\sum_{i \in I_u \cap I_v}(r_{ui} - \mu_u)(r_{vi} - \mu_v)}{\sqrt{\sum_{i \in I_u}(r_{ui} - \mu_u)^2}\sqrt{\sum_{i \in I_v}(r_{vi} - \mu_v)^2}} \tag{8}$$

où r_{ui} et r_{vi} sont les notes données respectivement par les utilisateurs u et v à l'item i, μ_u et μ_v sont les moyennes respectives des notes données par les utilisateurs u et v, I_u et I_v sont respectivement l'ensemble des items notés par u et par v. On ne prend en compte dans la somme du numérateur que les items qui ont été notés par les deux utilisateurs. On reconnait la formule du coefficient de corrélation.

Notre implémentation de cette formule se divise en deux fonctions. La première fonction permet de précalculer pour chaque utilisateur leur moyenne de notes μ_u et la racine carrée de la somme du carré des écarts à leur moyenne $\sqrt{\sum_i (r_{ui} - \mu_u)^2}$. Ces informations sont stockées dans deux listes afin d'optimiser le temps de calcul, puis la deuxième fonction les utilise pour implémenter

l'équation 7.

Il faut cependant introduire ici une correction. En effet, on ne calculera la similarité entre deux individus que si le nombre d'items qu'ils ont tous les deux noté est supérieur à un certain seuil à déterminer selon la taille du jeu de données. Si on ne prend pas en compte ce seuil, alors deux utilisateurs qui n'auront noté qu'un seul item en commun auront une corrélation égale à 1 alors qu'ils ne seront pas du tout similaires. Dans notre algorithme, nous avons choisit un seuil T=50 car un seuil inférieur fausse les similarités et qu'un seuil supérieur n'est pas adapté à notre jeu de données qui n'est pas très grand.

D'autre part, une seconde correction est à apporter lors de l'impémentation de cette formule. Si des utilisateurs ont accordé la même note à tous les films qu'ils ont regardé, alors on a $\sqrt{\sum_i (r_{ui} - \mu_u)^2} = 0$ ce qui pose problème pour le calcul de la similarité. Dans ce cas là on considère la similarité nulle, ce qui revient à supprimer l'utilisateur de notre jeu de données. Il s'agit donc d'une faille de l'algorithme des plus proches voisins qui ne peut pas inclure ces utilisateurs dans ses calculs de prédiction. A l'issue de cette étape, on obtient la matrice des similarités entre tous les utilisateurs.

Détermination du set des plus proches voisins

Une fois les similarités calculées, le set des plus proches voisins N_u peut être établi, il contient les utilisateurs dont les similarités avec l'utilisateur u considéré sont les plus élevées. Enfin, parmi les utilisateurs de ce set, on conserve ceux qui ont noté l'item i et on forme le set N_{ui}. La taille du set N_{ui} est à fixer en fonction de la taille du jeu de données, elle se situe généralement entre 20 et 60 utilisateurs car cela donne les meilleures performances.

Les scores MAE et RMSE ont été calculés sur un même set de test pour un nombre de proches voisins allant de 20 à 60.

On constate qu'avec le jeu de donnée Movielens, les erreurs MAE et RMSE sont minimales lorsque la taille du set des plus proches voisins est fixée à 23.

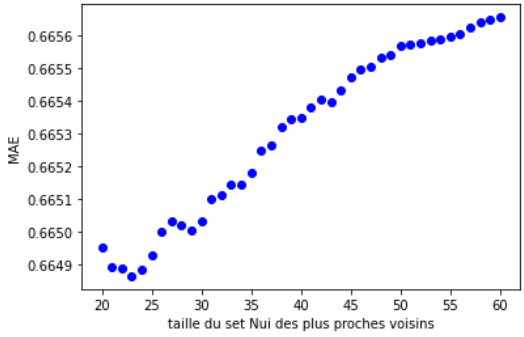


FIGURE 1 – graphique du score MAE obtenu en fonction de la taille du set N_{ui}

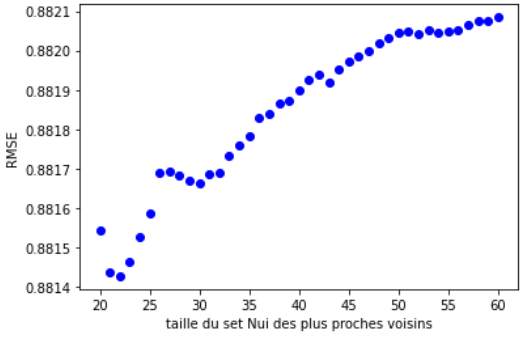


FIGURE 2 – graphique du score RMSE obtenu en fonction de la taille du set N_{ui}

Calcul du score

Une fois le set des plus proches voisins établi, on peut calculer la prédiction du score que l'utilisateur u donnerait à l'item i avec la formule suivante :

$$S(u,i) = \mu_u + \frac{\sum_{v \in N_{ui}} sim(u,v) * (r_{vi} - \mu_v)}{\sum_{v \in N_{ui}} |sim(u,v)|} \tag{9}$$

Le score est donc la moyenne des notations de l'utilisateur u à laquelle on ajoute la somme de la différence entre la note r_{vi} donnée par l'utilisateur voisin v à l'item et la moyenne μ_v des notations de v , pondérée par les similitudes de ces utilisateurs v avec l'utilisateur u considéré. Tout comme le prédicteur de base, il faut introduire une correction si le score calculé n'est pas compris dans l'intervalle [0.5, 5.0], on le force aux limites.

Utilisation du prédicteur de base si l'algorithme des plus proches voisins échoue

Il arrive que l'algorithme des plus proches voisins échoue. Par exemple, quand on construit le set des plus proches voisins N_{ui}, il se peut que ce set soit vide. En effet, si l'utilisateur u a noté très peu de films, le set de ses voisins N_u peut être très réduit, et donc si parmi les voisins v aucun n'a noté le film i considéré, le set N_{ui} sera vide. Cela pose problème pour le calcul du score puisqu'il fait intervenir une fraction dont le numérateur est la somme des valeurs absolues des similarités de l'utilisateur u avec ses plus proches voisins v appartenant au set N_{ui}. Pour ces éléments, nous faisons alors appel au prédicteur de base pour calculer le score.

V.2 Performance de l'algorithme sur le jeu de données Movielens

Enfin, on peut tester la fonction de score sur les 20% de données manquantes et comparer la prédiction aux valeurs réelles. Cette fonction de test calcule les scores MAE et RMSE obtenus

sur le jeu de données. Lorsque l'on teste l'algorithme sur les 20% de données manquantes, le score MAE obtenu est de 0.6648 et le score RMSE est de 0.8814. Ainsi, les scores MAE et RMSE obtenus avec l'algorithme des plus proches voisins sont meilleurs que ceux du prédicteur de base. Cela dit, les scores sont assez proches, l'algorithme des plus proches voisins n'a donc pas apporté une très grande amélioration. Cela peut s'expliquer par ses nombreuses failles. Premièrement, il ne permet pas de traiter les utilisateurs ayant donné la même note à tous les films. Ensuite, il est impacté par la taille du jeu de données, puisque l'on a 9742 films pour seulement 610 utilisateurs il est probable que chaque utilisateur ait peu de plus proches voisins dans son set N_{ui} et que donc la prédiction soit moins bonne ou faussée. A l'inverse, le prédicteur de base n'est pas sensible à ces perturbations et fournit une prédiction dans la plupart des cas.

V.3 Performance de l'algorithme sur le jeu de données Jester

En comparaison avec le prédicteur de base, l'algorithme des plus proches voisins se montre plus efficace que sur le jeu de données Jester, que sur le jeu de données Movielens. Les scores MAE et RMSE ont été calculés sur un même set de test pour un nombre de proches voisins allant de 20 à 60.

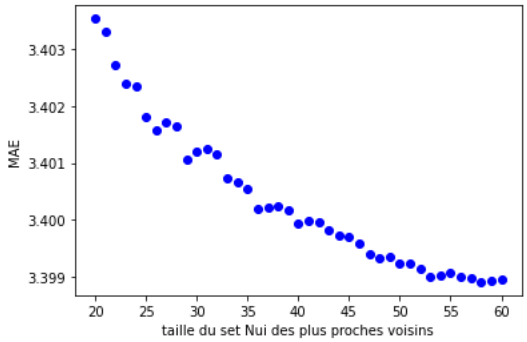


FIGURE 3 – graphique du score MAE obtenu en fonction de la taille du set N_{ui}

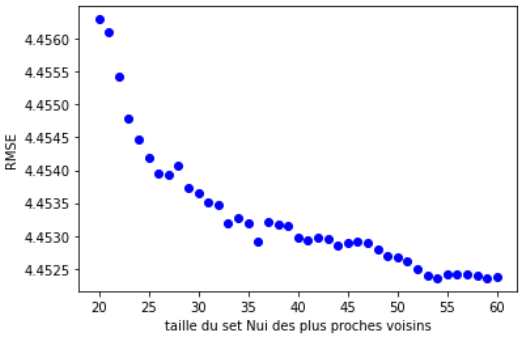


FIGURE 4 – graphique du score RMSE obtenu en fonction de la taille du set N_{ui}

On constate que les meilleurs scores sont obtenus pour une taille du set des plus proches voisins fixée à 58. Le score MAE est alors de 3.3964, et le score RMSE de 4.4682. Il y a une nette amélioration par rapport au prédicteur de base dont les résultats sur ce jeu de données sont un score MAE de 3.4834 et un score RMSE de 4.5683. Il est à noter que même si les scores sont beaucoup plus élevés que sur le jeu de données Movielens, cela ne signifie pas que la prédiction est moins bonne. En effet, sur le jeu de données Jester, la plage de notation est quatre fois plus étendue, puisque les notes vont de -10 à 10, alors que sur Movielens elles allaient de 0.5 à 5. Par

ailleurs, l'efficacité d'un algorithme de recommandation se jauge par rapport au score donné par le prédicteur de base. Enfin, l'algorithme des plus proches voisins montre davantage son efficacité sur ce jeu de données que sur le précédent puisqu'on voit que les scores décroissent globalement avec l'augmentation de la taille des plus proches voisins, tandis que sur le jeu de données précédent, la taille optimale pour le set des plus proches voisins était de 23 seulement, ensuite les scores augmentaient si on prenait en compte davantage de voisins. Or dans un jeu de données, plus on peut prendre en compte un nombre important de plus proches voisins, plus la prédiction est fiable car elle se base alors sur davantage de notations.

En conclusion, l'algorithme des plus proches voisins fournit une meilleure recommandation que celle du prédicteur de base, cependant il est moins robuste et ne peut pas fournir de prédictions dans plusieurs cas extrêmes, comme l'arrivée de nouveaux utilisateurs dans le jeu de données qui n'ont encore rien noté, ou des items qui n'ont pas reçu de notes. L'algorithme semble ne pas être adapté au jeu de données Movielens puisqu'il ne fournit qu'une petite amélioration par rapport au prédicteur de base. Cela peut s'expliquer par le fait que le jeu de données ne contienne que 610 utilisateurs contre 9742 items, la matrice R est ainsi très creuse, c'est-à-dire que l'on ne dispose pas de beaucoup de notes. En revanche, il fournit des scores nettement meilleurs par rapport au prédicteur de base sur le jeu de données Jester. Ce dernier contient en effet un nombre beaucoup plus important d'utilisateurs (7699) pour moins d'items (158). En revanche, 7% des blagues du jeu de données n'ont pas de notations ce qui impacte négativement l'algorithme des plus proches voisins.

VI Algorithme de descente de Gradient Stochastique ou Funk SVD

VI.1 Introduction

Un des principaux défauts de l'algorithme des plus proches voisins est qu'il détecte les relations par paires entre utilisateurs ou items sans tenir compte de structures plus larges dans les données.

En effet, nous pouvons imaginer qu'il existe des caractéristiques « cachées » qui déterminent comment un utilisateur évalue un item. Par exemple, plusieurs items peuvent partager un sujet commun ou bien deux utilisateurs donneront des notes élevées à un certain film s'ils aiment tous deux le genre du film ou le réalisateur. Il existe alors une relation entre la note et le contenu des films. Les évaluations des utilisateurs peuvent donc s'expliquer par leur intérêt pour des caractéristiques particulières appelées caractéristiques latentes. Ainsi, au lieu de modéliser les relations deux par deux entre les utilisateurs ou les items, les techniques de factorisation matricielle sont généralement plus efficaces car elles permettent de découvrir les caractéristiques latentes représentant les interactions entre les utilisateurs et les items.

Les systèmes de recommandation disposent de l'évaluation d'utilisateurs pour certains items du système. Leur but est de prédire comment les utilisateurs évalueraient les items qu'ils n'ont pas encore évalués, afin de pouvoir leur faire des recommandations. Par conséquent, si nous pouvons découvrir ces caractéristiques latentes (i.e. ce lien entre la note et le contenu des films qui n'apparaît pas explicitement dans les données), nous devrions être en mesure de prédire une évaluation pour un utilisateur et un item donné, car les caractéristiques associées à l'utilisateur devraient correspondre aux caractéristiques associées à l'item.

VI.2 Construction des matrices

Soit U l'ensemble des m utilisateurs et I l'ensemble des n items. On note R la matrice de dimension $m \times n$ contenant l'ensemble des notations que les utilisateurs ont attribué aux items. Cette matrice est en grande partie vide (notes égales à 0 ou à 99 selon le jeu de données utilisé) car tous les utilisateurs n'ont pas noté tous les items.

Considérons l'ensemble des K caractérisitiques latentes à déterminer, en supposant que $K < m$ et $K < n$ (il y a moins de caractéristiques que d'items ou d'utilisateurs). Le but est alors de calculer les matrices P (de dimension $m \times K$) et Q (de dimension $n \times K$) de sorte qu'elles approximent au mieux R :

$$R \approx P \times Q^T = \widehat{R}$$

Où $\widehat{R}$ est la matrice des notations prédites. Au contraire de la matrice des notations exactes R, la matrice des notations prédites $\widehat{R}$ est entièrement remplie. Les matrices ci-dessus peuvent se schématiser de la manière suivante :

$$R = \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,n} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ r_{m,1} & r_{m,2} & \cdots & r_{m,n} \end{pmatrix}$$ où certaines entrées de R sont vides (égales à 0 ou 99)

$$P = \begin{array}{c} \begin{matrix} f^1 & f^2 & \cdots & f^K \end{matrix} \\ \begin{pmatrix} p_{11} & \cdots & p_{1K} \\ \vdots & & \vdots \\ p_{m1} & \cdots & p_{mK} \end{pmatrix} \end{array} \begin{matrix} p^{1*} \\ p^{2*} \\ \vdots \\ p^{m*} \end{matrix} \qquad Q = \begin{array}{c} \begin{matrix} f^1 & f^2 & \cdots & f^K \end{matrix} \\ \begin{pmatrix} q_{11} & \cdots & q_{1K} \\ \vdots & & \vdots \\ q_{n1} & \cdots & q_{nK} \end{pmatrix} \end{array} \begin{matrix} q^{1*} \\ q^{2*} \\ \vdots \\ q^{n*} \end{matrix}$$

Avec $(p^{u*})_{1\leq u\leq m}$ et $(q^{i*})_{1\leq i\leq n}$ les vecteurs ligne de P et de Q. Ainsi, chaque ligne de P représente la force des associations entre un utilisateur et les K caractéristiques. De même, chaque ligne de Q représente la force des associations entre un item et les K caractéristiques. Pour calculer les coefficients de la matrice $\widehat{R}$ nous effectuons le produit scalaire :

$$\hat{r_{ui}} = p^{u*}q^{i} = \sum_{k=1}^{K} p_{uk}q_{ki}$$

où $\hat{r_{ui}}$ est la prédiction de la note de l'utilisateur u pour l'item i. Il faut à présent trouver les matrices P et Q et calculer à quel point le produit $P \times Q^T = \widehat{R}$ est différent de R. L'algorithme doit alors minimiser cette différence à chaque itération. Pour ce faire, nous utilisons la méthode de descente de gradient stochastique.

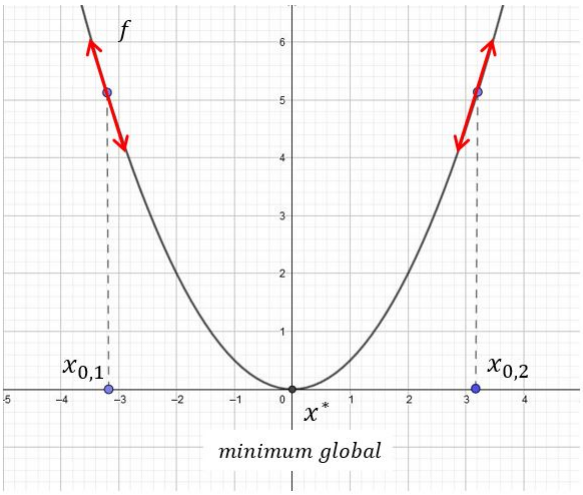
VI.3 Descente de Gradient

La méthode de descente de gradient permet de trouver le lieu du minimum x^* d'une fonction f différentiable. On choisit un point initial x_0 et on calcule récursivement :

$$\forall k \geq 0 \ : \ x_{k+1} \ = \ x_k \ - \ \alpha \frac{\partial f}{\partial x}[x_k], \quad \alpha > 0$$

où α est le taux d'apprentissage. La suite $(x_k)_{k\geq 0}$ converge alors vers la solution x^*.

Illustration :



La descente de gradient calcule la dérivée de la fonction f afin de savoir dans quelle direction choisir le prochain point sur lequel itérer pour qu'il se rapproche de la solution x^*. Sur l'illustration, la pente de la courbe est négative en $x_{0,1}$. On a alors $\alpha \frac{\partial f}{\partial x}[x_k] \leq 0$ soit $x_{k+1} \geq x_k$. En $x_{0,2}$, la pente est positive et donc $x_{k+1} \leq x_k$.
A chaque itération k, le point x_k se rapproche de x^*.

En général, la fonction f que nous cherchons à minimiser est une fonction de plusieurs variables. Nous utilisons alors la formule :

$$\forall k \geq 0 \ : \ x_{k+1} \ = \ x_k \ - \ \alpha \nabla f[x_k], \quad \alpha > 0$$

où ∇f représente le gradient de f.

Descente de Gradient en machine learning

La méthode de descente de gradient est une méthode itérative utilisée pour minimiser une fonction objectif. Elle connaît un très grand intérêt aujourd'hui, en particulier pour l'entraînement des réseaux de neurones.
Considérons $N+1$ données $(X_i, y_i)_{i=1:N}$ où les $X_i \in \mathbb{R}^l$ constituent l'entrée du réseau et les $y_i \in \mathbb{R}$ la sortie attendue. Le but est alors de trouver un modèle mathématique $F : \mathbb{R}^l \longrightarrow \mathbb{R}$ tel que $F(X_i) \approx y_i$ où $F(X_i)$ représente la sortie produite par notre modélisation. La fonction F dépend de paramètres $(a_1, ..., a_n)$ qui seront déterminés par la méthode de descente de gradient. Ce sont par exemple les poids dans le réseau de neurones. Nous définissons de plus l'erreur locale (erreur pour la donnée i) : $E_i = (y_i - F(X_i))^2$ qui est la différence entre la sortie attentue et la sortie produite par notre modèle. Elle est fonction des paramètres inconnus $(a_1, ..., a_n)$: $E_i(a_1, ..., a_n)$. L'erreur totale $E = \sum_{i=1}^{N} E_i(a_1, ..., a_n)$ est la somme des erreurs locales pour toutes nos données. La descente de gradient a pour objectif de minimiser E.

Descente de Gradient Classique Nous partons d'un point initial P_0 représentant une certaine valeur de nos paramètres inconnus $(a_1, ..., a_n)$: $P_0 = (a_1, ..., a_n)$. Puis, nous calculons par récurrence les points suivants avec la formule du gradient jusqu'à convergence de l'algorithme :

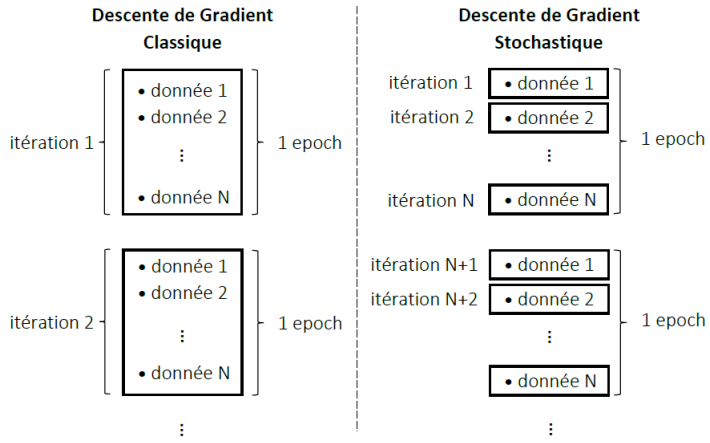
$$P_1 = P_0 - \alpha \nabla E(P_0)$$
$$P_2 = P_1 - \alpha \nabla E(P_1)$$
$$\dots$$

Or, $\nabla E(P_j) = \sum_{i=1}^{N} \nabla E_i(P_j)$ donc pour chaque itération, nous devons calculer $\nabla E(P_j)$ qui dépend des $N+1$ données. Cette méthode n'est donc pas réalisable dans le cas d'un très grand nombre de données puisqu'elle peut générer des problèmes mémoire ou des temps de calculs excessivement longs.

Descente de Gradient Stochastique La descente de gradient stochastique consiste à utiliser une seule donnée à chaque itération. Nous partons toujours d'un point initial $P_0 = (a_1, ..., a_n)$. Puis nous calculons les itérations suivantes :

$$P_1 = P_0 - \alpha \nabla E_2(P_0)$$
$$P_2 = P_1 - \alpha \nabla E_7(P_1)$$
$$\dots$$

La principale différence est que l'on se sert de l'erreur locale $E_i(P_j)$ et non de l'erreur globale $E(P_i)$ pour le calcul de l'itération suivante. A chaque itération, nous utilisons un E_i différent qui est sélectionné aléatoirement (dans notre exemple, c'est E_2 qui est utilisé puis E_7...). Une fois que tous les E_i - donc toutes les données - ont servi une fois (lorsque nous avons réalisé un "epoch"), nous faisons à nouveau un tour à travers toutes les données. Nous réitérons le processus jusqu'a convergence de l'arlgorithme. Le fait de choisir aléatoirement les E_i permet d'éviter que la solution ne converge vers un minimum local. C'est cette méthode que nous implémenterons pour l'algorithme Funk-SVD.



Le schéma ci-contre illustre la différence entre l'algorithme de descente de gradient classique et stochastique

Paramètre et critère d'arrêt En général, le taux d'apprentissage α est petit et égal à 0.1, 0.001 ou 0.0001. Le choix de α est crucial pour la convergence de l'algorithme. En effet, suivant sa valeur, la méthode du gradient peut diverger. De très petites valeurs de α assurent généralement une convergence, mais lente, tandis que de trop grandes valeurs de α font diverger l'algorithme. Le choix de ce paramètre sera ici effectué de manière empirique, en testant plusieurs valeurs. Par la suite, le critère d'arrêt retenu sera le nombre d'itérations à ne pas excéder.

L'algorithme implémenté ici est constitué de deux fichiers python. Le premier fichier (fonction *compute_Funk_SVD* détaillée en annexe) contient une fonction codant la descente de gradient stochastique. Cette fonction permet de construire la matrice $\widehat{R}$ contenant les prédictions des notes des utilisateurs pour les items. Le second fichier (nommé *Run_Funk_SVD.py* détaillé en annexe) est le code d'exécution de la fonction de descente de gradient stochastique. Il permet notamment de déterminer les paramètres optimaux et la performance de la descente de gradient stochastique.

Les sections suivantes présentent les équations mathématiques intervenant dans la fonction codant l'algorithme de descente de gradient stochastique.

VI.4 Formulation du problème

Dans notre cas, les paramètres inconnus "$(a_1, ..., a_n)$" que nous cherchons à déterminer sont représentés par les matrices P et Q. Plus précisément, ces paramètres sont l'ensemble des coefficients p_{uk} et q_{ki} de P et Q. Mettre à jour leur valeur revient donc à itérer sur l'ensemble des coefficients des deux matrices. Afin de constituer le point initial, nous commencerons par initialiser P et Q avec des valeurs aléatoires. Les erreurs "E_i" que nous cherchons alors à minimiser sont les erreurs élémentaires e_{ui} entre les coefficients (non nuls) r_{ui} de R et les coefficients $\hat{r_{ui}}$ de $\widehat{R}$:

$$e_{ui}^2 = (r_{ui} - \hat{r_{ui}})^2 \;=\; (r_{ui} - \sum_{k=1}^{K} p_{uk} q_{ki}\ )^2 \quad \forall 1 \leq u \leq m,\ \forall 1 \leq i \leq n$$

L'erreur au carré est utilisée car la différence peut être positive ou négative.

Il faut à présent calculer la dérivée de la fonction e_{ui}^2 par rapport aux variables p_{uk} et q_{ki}
$\forall\ 1 \leq k \leq K$:

$\frac{\partial}{\partial p_{uk}} e_{ui}^2 \;= 2(-\frac{\partial}{\partial p_{uk}} \sum_{k'=1}^{K} p_{uk'} q_{k'i}\ )(r_{ui} \;-\; \hat{r_{ui}})$ or, $\frac{\partial}{\partial p_{uk}} p_{uk'} q_{k'i} = \begin{cases} 0 & \text{si } k \neq k' \\ q_{ki} & \text{si } k = k' \end{cases}$

donc $\frac{\partial}{\partial p_{uk}} e_{ui}^2 = -2(q_{ki})(r_{ui} - \hat{r_{ui}}) = -2e_{ui}q_{ki}$

et $\frac{\partial}{\partial q_{ki}} e_{ui}^2 = -2(p_{uk})(r_{ui} - \hat{r_{ui}}) = -2e_{ui}p_{uk}$

Nous pouvons alors écrire l'expression pour l'itération suivante :

$$p_{uk}^{new} = p_{uk} - \alpha \frac{\partial}{\partial p_{uk}} e_{ui}^2 = p_{uk} + 2\alpha e_{ui} q_{ki}$$

$$q_{ki}^{new} = q_{ki} - \alpha \frac{\partial}{\partial q_{ki}} e_{ui}^2 = q_{ki} + 2\alpha e_{ui} p_{uk}$$

VI.5 Régularisation du problème

L'ajout d'un terme de régulatisation améliore les performances de l'algorithme notamment si le problème initial est mal conditionné. Cela permet aussi d'éviter le surajustement (ou sur-aprentissage) qui survient lorsqu'un modèle mathématique contient plus de paramètres que ne peuvent le justifier les données, c'est-à-dire lorsque l'algorithme apprend à partir des données mais aussi à partir de motifs qui ne sont pas liés au problème, comme du bruit. Dans notre cas, le terme de régularisation β pénalise les valeurs extrêmes de p_{uk} et q_{ki}. La fonction à minimiser devient donc :

$$e_{ui}^2 = (r_{ui} - \hat{r_{ui}})^2 + \frac{\beta}{2}(\|p^{u*}\|^2 + \|q^{i*}\|^2) = (r_{ui} - \sum_{k=1}^{K} p_{uk}q_{ki})^2 + \frac{\beta}{2}(\|p^{u*}\|^2 + \|q^{i*}\|^2)$$

avec β valant 0.02 en général. Nous devons donc recalculer la dérivée de e_{ui}^2 avec cette nouvelle expression :

$$\frac{\partial}{\partial p_{uk}} e_{ui}^2 = -2e_{ij}q_{kj} + \frac{\beta}{2}\frac{\partial}{\partial p_{uk}}(\|p^{u*}\|^2 + \|q^{i*}\|^2) \qquad \forall\ 1 \leq k \leq K$$

or les normes vectorielles s'écrivent :

$$\|p^{u*}\|^2 = \sum_{k'=1}^{K} (p_{uk'})^2 \text{ et } \|q^{i*}\|^2 = \sum_{k'=1}^{K} (q_{ik'})^2$$

donc $\frac{\partial}{\partial p_{uk}}\|p^{u*}\|^2 = \begin{cases} 2p_{uk} & \text{si k' = k} \\ 0 & \text{sinon} \end{cases}$ et $\frac{\partial}{\partial p_{uk}}\|q^{i*}\|^2 = 0$

d'où :

$$\frac{\partial}{\partial p_{uk}} e_{ui}^2 = -2e_{ui}q_{ki} + \beta p_{uk}$$

Et de la même façon :

$$\frac{\partial}{\partial q_{ki}} e_{ui}^2 = -2e_{ui}p_{uk} + \beta q_{ki}$$

Les expressions pour l'itération suivante deviennent :

$$p_{uk}^{new} = p_{uk} - \alpha \frac{\partial}{\partial p_{uk}} e_{ui}^2 = p_{uk} + \alpha(2e_{ui}q_{ki} - \beta p_{uk})$$

$$q_{ki}^{new} = q_{ki} - \alpha \frac{\partial}{\partial q_{ki}} e_{ui}^2 = q_{ki} + \alpha(2e_{ui}p_{uk} - \beta q_{ki})$$

VI.6 Ajout des termes de biais

Dans les calculs précédents, nous n'avons pas considéré l'existence de biais pouvant contribuer aux notations des utilisateurs pour les items. En effet, un utilisateur peut avoir tendance à donner des notes plus élevées ou plus faibles que la moyenne. De la même façon, un item peut être mieux ou moins bien noté que les autres items. Nous introduisons donc les vecteurs bu (de taille m) et bi (de taille n) représentant les biais des utilisateurs et des items :
$bu = (bu_1, bu_2, ..., bu_m)$ indique si en moyenne, un utilisateur a tendance noter haut ou faible.
$bi = (bi_1, bi_2, ..., bi_n)$ indique en moyenne, un item est mieux ou moins bien noté que la moyenne.
Nous introduisons également la moyenne globale μ représentant la moyenne des notes attribuées.
Par conséquent, la note prédite de l'utilisateur u pour la l'item i devient :

$$\hat{r_{ui}} = \mu + bu_u + bi_i + \sum_{k=1}^{K} p_{uk}q_{ki}$$

avec $\mu = \sum_{\substack{r_{ui} \\ r_{ui} \neq 0}} \frac{r_{ui}}{\#\{r_{ui}, r_{ui} \neq 0\}}$

Les biais bu et bi sont mis à jour à chaque itération de la même façon que les coefficients p_{uk} et q_{ki} et permettent à l'algorithme de converger plus rapidement. L'expression de e_{ui}^2 est donc à nouveau transformée et s'exprime de la manière suivante :

$$e_{ui}^2 = (r_{ui} - \hat{r_{ui}})^2 \ = \ (r_{ui} - \mu - bu_u - bi_i - \sum_{k=1}^{K} p_{uk}q_{ki} \ )^2$$

On ajoute à cette expression les termes de régularisation :

$$\begin{aligned} e_{ui}^2 = (r_{ui} - \mu - bu_u - bi_i - \sum_{k=1}^{K} p_{uk}q_{ki} \ )^2 \\ + \frac{\beta}{2}(\|p^{u*}\|^2 + \|q^{i*}\|^2) \\ + \frac{\beta}{2}(\|bu\|^2 + \|bi\|^2) \end{aligned} \tag{10}$$

où $\|bu\|^2 = \sum_{u'=1}^{m} \ bu_{u'}^2$ et $\|bi\|^2 = \sum_{i'=1}^{n} \ bi_{i'}^2$

Il faut donc à présent dériver e_{ui}^2 par rapport aux quatre variables p_{uk}, q_{ki}, bu_u et bi_i. La dérivée de e_{ui}^2 par rapport à p_{uk} et q_{ki} reste inchangée et donc les expressions de p_{uk}^{new} et q_{ki}^{new} sont les mêmes que celles déterminées ci-dessus. La dérivée de e_{ui}^2 par rapport à bu_u se calcule comme suit : $\forall\ 1 \leq k \leq K$:

$$\begin{aligned}\frac{\partial}{\partial bu_u} e_{ui}^2 = \frac{\partial}{\partial bu_u}(r_{ui} - \mu - bu_u - bi_i - \sum_{k=1}^{K} p_{uk}q_{ki}\ )^2 \\ + \frac{\beta}{2}\frac{\partial}{\partial bu_u}(\|p^{u*}\|^2 + \|q^{i*}\|^2) \\ + \frac{\beta}{2}\frac{\partial}{\partial bu_u}(\|bu\|^2 + \|bi\|^2)\end{aligned} \tag{11}$$

$$\begin{aligned}\Leftrightarrow \frac{\partial}{\partial bu_u} e_{ui}^2 = -2(r_{ui} - \mu - bu_u - bi_i - \sum_{k=1}^{K} p_{uk}q_{ki}\ ) \\ + 0 \\ + \frac{\beta}{2}(2\ bu_u)\end{aligned} \tag{12}$$

$$\Leftrightarrow \frac{\partial}{\partial bu_u} e_{ui}^2 = -2e_{ui} + \beta bu_u$$

De manière équivalente on obtient :

$$\frac{\partial}{\partial bi_i} e_{ui}^2 = -2e_{ui} + \beta bi_i$$

Nous pouvons donc écrire les expressions pour l'itération suivante :

$$bu_u^{new} = bu_u - \alpha\frac{\partial}{\partial bu_u} e_{ui}^2 = bu_u + \alpha(2e_{ui} - \beta bu_u)$$

$$bi_i^{new} = bi_i - \alpha\frac{\partial}{\partial bi_i} e_{ui}^2 = bi_i + \alpha(2e_{ui} - \beta bi_i)$$

Dans notre code, nous considérerons que le coefficient 2 est compris dans α.

L'algorithme Funk-SVD entraîne donc P et Q, afin que $P \times Q^T = \widehat{R}$ se rapproche à chaque itération du résultat exact R. Pour vérifier que les matrices P et Q sont "bien entraînées", nous éliminons un certain nombre de notations connues r_{ui}^{elim} de R. Puis, nous comparons si les entrées correspondantes $r_{ui}^{\hat{}}$ de la matrice prédite $\widehat{R}$ sont similaires. Si $r_{ui}^{elim} \approx r_{ui}^{\hat{}}$ alors P et Q sont performantes et nous pouvons donc les utiliser pour formuler des recommandations.

VI.7 Implémentation

Fonction implémentant la descente de gradient stochastique

Nous disposons à présent des équations nécessaires pour implémenter la fonction de descente de gradient stochastique nommée $compute\_Funk\_SVD$. Cette fonction prend en argument la matrice d'entrainement R, le nombre de caractéristiques latentes K, le taux d'apprentissage α, le paramètre de régularisation β et le nombre d'epochs à réaliser nb_epochs.
Comme évoqué précédemment, nous commençons par initialiser les matrices P et Q par des valeurs aléatoires. Les vecteurs de biais bu et bi sont initialisés à zéro et la moyenne μ avec la formule présentée en amont.
Un epoch de descente de gradient consiste à itérer une fois sur toutes les données. Nous construisons donc une liste de tuples (utilisateur u, item i, note de u pour i) : $\{(u, i, r_{ui}), r_{ui} \neq 0\}$ sur laquelle itérer. C'est la fonction $one\_step\_stochastic$ qui effectue un epoch : pour chaque tuple de la liste, elle met à jour les valeurs de bu_u, bi_i, p_{uk} et q_{ki} Au total nous effectuerons nb_epochs epochs.
Avant chaque epoch, l'ensemble des tuples $\{(u, i, r_{ui}), r_{ui} \neq 0\}$ est mélangé ce qui revient à choisir une valeur E_i au hasard (notation faisant référence à la section VI.3). Enfin, la fonction $total\_error$ permet de calculer à chaque epoch l'erreur totale E (cf. section VI.3). Si l'on observe que l'erreur totale diminue à chaque itération, cela permet de vérifier que l'algorithme converge.

Détermination des paramètres optimaux

Pour les quatre jeux de données utilisés, le paramètre de régularisation β est fixé à 0.02. Par la suite, 10 epoch ($nb\_epochs = 10$) donne des résultats satisfaisants. Le nombre de caractéristiques latentes K est choisi comme étant le nombre de valeurs singulières dominantes de la matrice d'entrainement R. Ce nombre est déterminé empiriquement à l'aide des graphiques ci-dessous.

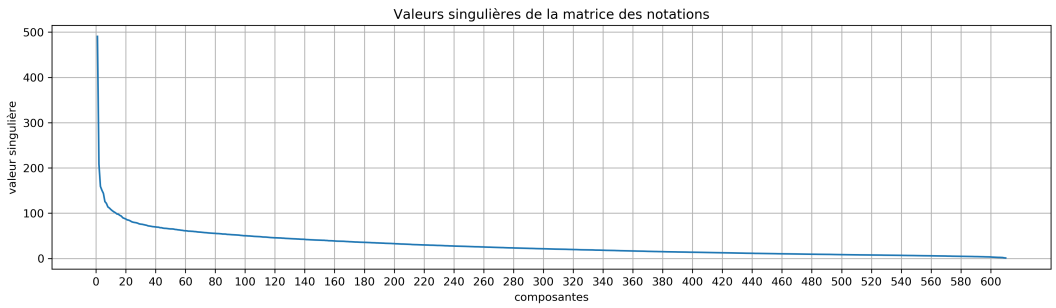


FIGURE 5 – Valeurs singulières de la matrice R du jeu de données Movielens

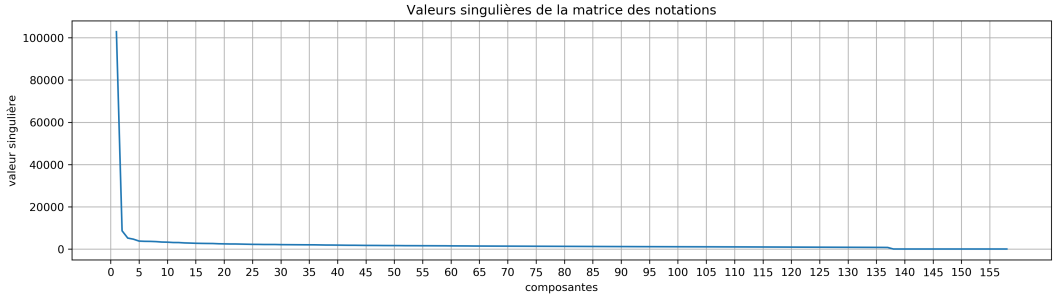


FIGURE 6 – Valeurs singulières de la matrice R du jeu de données Jester Dataset4

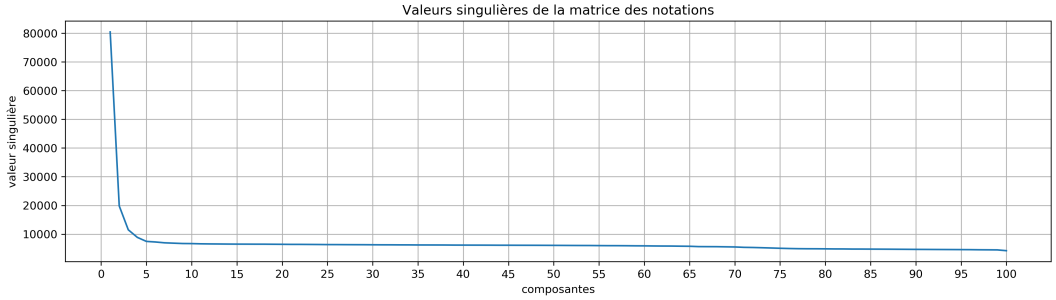


FIGURE 7 – Valeurs singulières de la matrice R du jeu de données Jester Dataset11

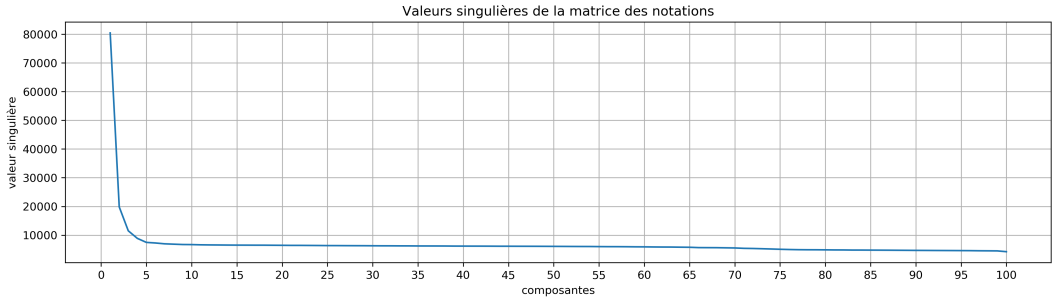


FIGURE 8 – Valeurs singulières de la matrice R du jeu de données Jester Dataset12

Pour le jeu de données Movielens, nous prenons $K = 200$ valeurs singulières dominantes. Pour les jeux de données Jester Dataset4, Jester Dataset11 et Jester Dataset12, nous prenons respectivement $K = 10$, $K = 15$ et $K = 15$.

Il nous faut à présent déterminer le paramètre de régularisation α. Comme expliqué précedemment, ce paramètre influe grandement sur la convergence de l'algorithme et sera ici déterminé de manière empirique. Pour les petits jeux de données MovieLens et Jester Dataset4, nous testons plusieurs valeurs de α afin de déterminer celle donnant l'erreur totale la plus faible après 10 epochs.

taux d'apprentissage

itérations	0.09	0.095	0.01	0.015	0.02	0.025
0	245,766	246,377	260,579	256,936	254,611	252,716
1	232,344	230,153	254,073	250,465	248,112	246,341
2	190,612	181,779	250,145	246,588	244,397	242,821
3	136,858	127,173	247,444	244,068	241,683	239,806
4	94,9095	88,9541	245,446	241,904	239,468	236,959
5	69,8959	65,74	243,726	240,201	237,26	232,666
6	55,7334	51,9661	242,402	238,508	234,415	225,278
7	47,2373	44,9615	241,094	236,993	229,413	214,356
8	41,4046	41,0668	239,954	234,543	222,383	200,801
9	39,8309	38,3428	238,839	231,336	213,07	185,7

FIGURE 9 – Erreur totale pour différentes valeurs de α pour le jeu de données MovieLens

taux d'apprentissage

itérations	0.001	0.0015	0.002	0.0025	0.003	0.0035	0.004	0.0045	0.005	0.0055	0.006
0	1434,09	1424,07	1415,94	1408,56	1401,82	1394,96	1388,73	1376,73	1350,81	7105,49	
1	1415,61	1402,05	1390,08	1378,81	1362,45	1312,14	1417,58	nan	nan	nan	nan
2	1401,88	1384,88	1367,75	1342,67	1261,77	1228,51	nan	nan	nan	nan	nan
3	1390,31	1368,92	1337,17	1261,9	1171,79	1119,96	nan	nan	nan	nan	nan
4	1379,71	1350,91	1274,02	1189,77	1109,29	1045,53	nan	nan	nan	nan	nan
5	1369,58	1323,95	1199,34	1137,51	1054,26	992,671	nan	nan	nan	nan	nan
6	1358,97	1278,82	1150,34	1085,16	1006,46	952,283	nan	nan	nan	nan	nan
7	1346,55	1220,58	1111,54	1039,1	967,301	924,127	nan	nan	nan	nan	nan
8	1330,53	1171,7	1075,36	999,481	938,428	904,372	nan	nan	nan	nan	nan
9	1308,35	1134,02	1042,28	966,309	918,205	889,401	nan	nan	nan	nan	nan

FIGURE 10 – Erreur totale pour différentes valeurs de α pour le jeu de données Jester Dataset4

Les meilleurs résultats sont obtenus avec $\alpha = 0.095$ pour MovieLens et avec $\alpha = 0.0035$ pour Jester Dataset4. Pour les jeux de données plus conséquents comme Jester Dataset11 et Jester Dataset12, tester plusieurs valeurs de α serait trop long. La valeur $\alpha = 0.0001$ pour Dataset11 et Dataset12 donne des résultats satisfaisants.

VI.8 Performance de l'algorithme de Funk-SVD

Le tableau ci-dessous présente les résultats du calcul de la MAE et de la RMSE pour chaque jeu de données.

Jeu de données	MAE	RMSE
MovieLens	0.7068	0.9267
Jester Dataset4	3.5648	4.8137
Jester Dataset11	4.0747	4.9225
Jester Dataset12	4.1298	4.9890

Les valeurs de MAE et RMSE ne sont pas comparables entre les différents jeux de données. Cependant, nous pouvons supposer que le score pour le jeu de données MovieLens est meilleur que celui des jeux de données Jester car l'amplitude des notations est plus faible. Il y a donc

moins de risque d'erreur. Étonnamment, nous constatons que l'algorithme de Funk-SVD possède une moins bonne performance que l'algorithme du prédicteur de base ou celui des plus proches voisins. Ceci peut s'expliquer par le fait que les algorithmes de factorisation matricielle donnent de meilleurs résultats pour une très grande quantité de données. En effet, à ce stade, l'espace de stockage devient insuffisant pour les algorithmes basés sur la mémoire, et il y a suffisamment de données pour que les caractéristiques latentes deviennent très apparentes.

Dans les quatre bases de données considérées, nous pouvons observer que certains utilisateurs n'ont pas noté d'item et que certains items n'ont aucune note attribuée. Nous supprimons alors de la matrice des notations R les lignes et les colonnes correspondant à ces cas particuliers. En conservant les mêmes valeurs que précédemment pour les paramètres K, α, β et nb_epochs, nous obtenons les résultats suivants :

Jeu de données	MAE	RMSE
MovieLens	0.6620	0.8660
Jester Dataset4	3.5413	4.7977
Jester Dataset11	4.0712	4.9198
Jester Dataset12	4.1334	4.9938

Nous observons que l'algorithme de Funk-SVD a une meilleure performance lorsque nous supprimons les utilisateurs n'ayant pas noté d'item et les items n'ayant pas de note. En effet, pour les trois premiers jeux de données (MovieLens, Jester Dataset4 et Jester Dataset11), les valeurs de MAE et RMSE sont plus faibles que precédemment. La différence de performance est d'autant plus grande que le jeu de données est de petite taille. Il s'agit donc d'un défaut de l'algorithme de Funk-SVD, qui donne de moins bons résultats lorsqu'il y a des notations manquantes, ce qui est souvent le cas en réalité. Au contraire, pour le jeu de données Jester Dataset12, la performance est moins bonne que précédemment. Nous pouvons donc en déduire que les lacunes de l'algorithme concernant les notations manquantes sont compensées par la grande quantité de données présentes dans Jester Dataset12.

VII Algorithme des moindres carrés alternés

VII.1 Introduction

Une seconde méthode de factorisation matricielle consiste à utiliser l'algorithme des moindres carrés alternés (ALS). L'algorithme des moindres carrés alternés présente 3 principaux avantages. Tout d'abord, il s'avère que contrairement à l'algorithme des plus proches voisins ou encore le prédicteur de base, l'ALS est, comme la Funk SVD, adapté aux grands jeux de données. Nous verrons d'ailleurs par la suite que les résultats du RMSE et du MAE sont meilleurs avec l'ALS pour les grandes bases de données telles que Jester Dataset11. Ensuite, cet algorithme est aussi approprié dans le cas où le jeu de données contient très peu de notes par rapport à la quantité d'utilisateurs et d'items. Enfin, l'ALS est aussi un algorithme qui s'implémente assez facilement et propose des résultats en peu de temps.

VII.2 Construction des matrices

On rappelle ici la définition de notre matrice utilisateur-item:

$$R = \begin{pmatrix} r_{1,1} & \cdots & r_{1,n} \\ \cdots & r_{u,i} & \cdots \\ r_{m,1} & \cdots & r_{m,n} \end{pmatrix}$$

où $r_{u,i}$ représente la note du i^e film donnée par le u^e utilisateur et les entiers n et m sont respectivement le nombre de films et le nombre d'utilisateurs.

L'objectif de cette partie est d'estimer les valeurs manquantes de la matrice R en se basant sur les notes que nous avons à notre disposition. L'idée ici est de réaliser une approximation de la matrice R par une matrice à l'aide d'une factorisation matricielle.

La factorisation matricielle s'écrit alors : $\tilde{\mathrm{R}} = U^T M$ où $\tilde{\mathrm{R}}$ est une approximation de notre matrice utilisateur-item R et dont aucune des entrées n'est vide. Les matrices $U \in \mathcal{M}_{K\times m}$ et $M \in \mathcal{M}_{K\times n}$ (avec K qui représente le nombre de caractéristiques latentes, joue le même rôle que pour la Funk SVD) sont définies comme les matrices utilisateurs et items puisqu'elles contiennent les vecteurs colonnes u_u et m_i appelés *vecteurs latents*. Ces vecteurs sont associés respectivement à l'utilisateur u et au film i. Nous pouvons représenter les matrices U et M de la façon suivante :

$$U = \begin{array}{c} \begin{matrix} u_1 & \dots & u_u & \dots & u_m \end{matrix} \\ \begin{pmatrix} \vdots & & \vdots & & \vdots \end{pmatrix} \end{array} \text{ et } M = \begin{array}{c} \begin{matrix} m_1 & \dots & m_i & \dots & m_n \end{matrix} \\ \begin{pmatrix} \vdots & & \vdots & & \vdots \end{pmatrix} \end{array}$$

On souhaite avoir une approximation de R la plus fidèle possible i.e on souhaite minimiser l'écart entre r_{ui} et le produit scalaire $< u_u, m_i >$ et ce, $\forall u, i$. On introduit alors la *fonction perte* qui est la somme de tous ces écarts :

$$f(U,M) = \sum_{(u,i)\in I} \left(r_{ui} - u_u^T m_i\right)^2 + \lambda\left(\sum_u m_u \| u_u\|^2 + \sum_i n_i \| m_i\|^2\right)$$

Le premier terme correspond à la fonction coût et le second au terme de régularisation de Tikhonov afin d'éviter l'*overfitting* (surajustement).

VII.3 Existence du minimum de la fonction perte

Ainsi, nous nous sommes ramenés à un problème de minimisation de moindres carrés puisque nous voulons déterminer les matrices U et M pour lesquelles la fonction perte est minimale. Notre fonction dépend alors de deux variables U et M. Cependant, pour résoudre ce problème, nous considérerons alternativement que l'un des deux facteurs matriciels est nul.

On commence d'abord par s'assurer que notre problème de minimisation admet bien une solution lorsque M ou U est constant. On suppose par exemple, la fonction f constante selon la direction de M. On vérifie alors que notre fonction objectif est $\mathcal{C}^\infty$ comme somme de fonctions $\mathcal{C}^\infty$. Il nous faut ensuite montrer le caractère coercif de f.

$$f(U,M) = \sum_{(u,i)\in I} \left(r_{ui} - u_u^T m_i\right)^2 + \lambda\left(\sum_u m_u \| u_u\|^2 + \sum_i n_i \| m_i\|^2\right) \geq \lambda \sum_u m_u \| u_u\|^2 = g(\| u_u\|)$$

Avec : $\begin{array}{ccccc} g & : & \mathbb{R}^+ & \to & \mathbb{R} \\ & & x & \mapsto & g(x) \end{array}$ telle que $\lim_{x\to+\infty} g(x) = +\infty$. Ainsi la fonction f est bien coercive selon la direction des u_u. Finalement, la fonction f admet bien un minimum dans la direction des u_u. Avec un raisonnement similaire on montre que la fonction f admet également un minimum global selon les m_i.

Nous montrerons d'ailleurs dans la section VII.4, que ces minima sont uniques.

VII.4 Calcul du minimum de la fonction perte

Pour obtenir le couple (U, M) qui minimise f, nous devons chercher les points critiques qui sont alors les seuls candidats. Pour ce faire, on étudie dans chaque direction des u_u puis des m_i en "quel point" le gradient s'annule. On considère d'abord la direction selon les u_u en supposant alors les m_i constants :

$$\begin{aligned}
&\frac{1}{2}\frac{\partial f}{\partial u_{ku}}(U,M) = 0 \qquad \forall u,k \\
\Longrightarrow\ &\frac{1}{2}\frac{\partial}{\partial u_{ku}}\Big[\sum_{(u',i)\in I}\big(r_{u'i} - u_{u'}^T m_i\big)^2 + \lambda\big(\sum_{u'} m_{u'}\| u_{u'}\|^2 + \sum_i n_i\| m_i\|^2\big)\Big] = 0 \qquad \forall u,k \\
\Longrightarrow\ &\sum_{i\in I_u}\big(r_{ui} - u_u^T m_i\big)\frac{\partial}{\partial u_{ku}}\big[r_{ui} - u_u^T m_i\big] + \frac{\lambda}{2} m_u \frac{\partial}{\partial u_{ku}}\big[\sum_{k'=1}^{K} u_{k'u}^2\big] = 0 \qquad \forall u,k
\end{aligned}$$

Avec I_u un ensemble contenant les identifiants des films que l'utilisateur u a notés (il possède donc m_u éléments).
Or,

$$\frac{\partial}{\partial u_{ku}}\big(r_{ui} - u_u^T m_i\big) = \frac{\partial}{\partial u_{ku}}\big(-\sum_{k'=1}^{K} u_{uk'} m_{k'i}\big) = \begin{cases} 0 & si \quad k' \neq k \\ -m_{ki} & si \quad k' = k \end{cases}$$

De même,

$$\frac{\partial}{\partial u_{ku}}\big[\sum_{k'=1}^{K} u_{k'u}^2\big] = \frac{\partial}{\partial u_{ku}}\big(u_{1u}^2 + u_{2u}^2 + ... + u_{k'u}^2 + ... + u_{Ku}^2\big) = \begin{cases} 0 & si \quad k' \neq k \\ 2u_{ku} & si \quad k' = k \end{cases}$$

D'où,

$$\frac{1}{2}\frac{\partial f}{\partial u_{ku}}(U,M) = 0 \implies \sum_{i\in I_u}\big(u_u^T m_i - r_{ui}\big)m_{ki} + \lambda m_u u_{ku} = 0 \qquad \forall u,k$$

Par symétrie du produit scalaire, on obtient finalement :

$$\underbrace{\sum_{i\in I_u}\big(m_{ki} m_i^T u_u\big)}_{A} + \underbrace{\lambda m_u u_{ku}}_{B} = \underbrace{\sum_{i\in I_u} m_{ki} r_{ui}}_{C} \qquad \forall u,k$$

Notons que cette expression peut se mettre sous forme matricielle. En effet :

- *Terme A :* le système précédent est vérifié pour tout $i \in [\![1,m]\!], k \in [\![1,K]\!]$ i.e :

$\begin{cases} m_{1i}m_i^T u_u \\ m_{2i}m_i^T u_u \\ \dots \\ m_{Ki}m_i^T u_u \end{cases}$, avec $i \in I_u$. Ce système se réécrit de manière équivalente par : $M_{Iu}M_{I_u}^T u_u$ où M_{Iu} correspond à la sous-matrice de M où les colonnes des items i it_i (cf en rouge sur l'illustration ci-dessous) notés par u ont été sélectionnées :

$$M_{Iu} = \begin{array}{c} \begin{matrix} it_1 & & it_i & & it_{mu} \end{matrix} \\ \begin{pmatrix} \vdots & & \vdots & & \vdots \\ \vdots & \dots & \vdots & \dots & \vdots \\ \vdots & & \vdots & & \vdots \end{pmatrix} \end{array}$$

Ainsi $M_{Iu} \in \mathcal{M}_{K \times m_u}$.

- *Terme B :* de même, le système précédent est vérifié pour tout $u \in [\![1, m]\!], k \in [\![1, K]\!]$ i.e :

$$\begin{cases} \lambda & m_u u_{1u} \\ & \dots \\ \lambda & m_u u_{ku} \\ & \dots \\ \lambda & m_u u_{Ku} \end{cases} \text{ qui se réécrit : } \quad \lambda m_u \begin{pmatrix} 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \end{pmatrix} \begin{pmatrix} u_{1u} \\ u_{2u} \\ \dots \\ u_{Ku} \end{pmatrix} = \lambda m_u E u_u$$

où E est la matrice identité de taille $K \times K$ et u_u le vecteur latent associé à l'utilisateur u et de taille K.

- *Terme C :* De même on a $\sum_{i \in I_u} m_{ki} r_{ui}$ pour tout $u \in [\![1, m]\!], k \in [\![1, K]\!]$. En d'autres termes chaque composante du vecteur latent m_i associé à l'item i est multiplié par la notation que lui a attribué l'utilisateur u. On a donc le produit matrice-vecteur suivant :

$$\begin{array}{c} \begin{matrix} it_1 & & it_i & & it_{mu} \end{matrix} \\ \begin{pmatrix} \vdots & & \vdots & & \vdots \\ \vdots & \dots & \vdots & \dots & \vdots \\ \vdots & & \vdots & & \vdots \end{pmatrix} \end{array} \begin{pmatrix} r_{u,it_1} \\ \dots \\ r_{u,it_i} \\ \dots \\ r_{u,it_{mu}} \end{pmatrix} = M_{Iu} R(u, I_u)^T = V_u \quad \text{avec } R(u, I_u)^T \in \mathbb{R}^{m_u}$$

Finalement la matrice U est obtenue en résolvant $A_u u_u = V_u$ avec $A_u = M_{Iu}M_{I_u}^T + \lambda m_u E$

et $V_u = M_{Iu}R(u, I_u)^T$ pour chacune de ses colonnes u_u.

En suivant le même raisonnement on obtient un résultat similaire pour la matrice M i.e pour chaque colonne m_i, on résout le système suivant :

$$A_i m_i = V_i$$

Avec $A_i = U_{Ii}U_{I_i}^T + \lambda n_i E$ et $V_i = U_{Ii}R(I_i, i)$ avec I_i l'ensemble contenant les identifiants des utilisateurs ayant noté le film i (il possède donc n_i éléments). On définit alors U_{Ii} la sous matrice de U dont les colonnes u telles que $u \in I_i$ sont sélectionnées. Enfin, $R(I_i, i)$ est le vecteur colonne où chaque ligne $u \in I_i$ de la i^e colonne de R est sélectionnée. En d'autres termes, ce vecteur contient les notes attribuées à i par les utilisateurs u.

L'algorithme des moindres carrés alternés consistera à itérer ces opérations jusqu'à ce que l'écart entre le RMSE de l'itération précédente et celle courante soit plus petit qu'un certain seuil s_α (ici $s_\alpha = 0.0001$).

VII.5 Unicité du minimum de la fonction perte

Avec tous les calculs réalisés dans la section précédente, nous sommes en mesure de montrer que les minima trouvés dans chacune des directions u_u et m_i sont uniques. Il nous faut alors montrer la stricte convexité de la fonction perte selon u_u à m_i constant ou inversement. Pour ce faire, calculons la Hessienne $Hess_f$ de notre fonction perte en considérant par exemple les m_i constants puis montrons que ses valeurs propres appartiennent effectivement à $\mathbb{R}^{+*}$.

D'après la section précédente on a : $\frac{\partial f}{\partial u_u}(U, M) = \sum_{i \in I_u} \left(u_u^T m_i - r_{ui}\right) m_{ki} + \lambda m_u u_{ku}$. En dérivant une seconde fois selon u_l on a alors : $\frac{\partial^2 f}{\partial u_l \partial u_u}(U, M) = \sum_{i \in I_l} m_{ki}^2 + \lambda m_l, \forall k$. Ainsi pour tous les éléments de la Hessienne tels que $u \neq l$ alors la dérivée seconde croisée vaut 0. La Hessienne est donc une matrice diagonale dont les éléments diagonaux sont égaux à $\sum_{i \in I_l} m_{ki}^2 + \lambda m_l$ qui sont strictement positifs. En fait, ces réels sont les valeurs propres de la Hessienne de f et ainsi on a bien $Hess_f > 0$. Ainsi le minimum est bien unique selon u_u. Pareillement, le minimum selon les m_i est aussi unique.

VII.6 Implémentation de l'algorithme

Fonction implémentant l'algorithme des moindres carrés Nous disposons désormais de toutes les équations nécessaires à l'implémentation de notre algorithme. Cette fonction nommée *Alternating_Least_Square* prend en argument 3 paramètres :

— La matrice d'entraînement *R_train*

— Le paramètre de régularisation *param_reg*

— Le nombre de caractéristiques latentes K

— Le nombre maximal d'itérations à effectuer it_max

Cette fonction se divise essentiellement en 3 étapes : la première consiste à initialiser les matrices U et M. La matrice U est initialisée par des valeurs nulles pour tous ses coefficients alors que la matrice M est initialisée par des petites valeurs aléatoires (comprises entre 0 et 1) et dont la première ligne comporte la moyenne des notes obtenues par chaque film. La deuxième étape consiste à mettre à jour la matrice U en considérant M constante et inversement. La troisième étape consiste à mettre à jour la matrice M en considérant la matrice U constante. Les deuxièmes et troisièmes étapes sont répétées en boucle jusqu'à ce que le.s critère.s d'arrêt soi.en.t rempli.s (cf. sections suivantes).

Suppression des films non notés Concernant le jeu de données $Movilens$, après quelques exécutions de notre programme, nous avons pu constater une erreur systématique $singular\_matrix$. Cette erreur est due au fait que certains films n'ont pas été notés et les colonnes de la matrice correspondante ne contenaient donc que des 0 ce qui pouvait poser problème lors de la résolution des systèmes $A_u u_u = V_u$ et $A_i m_i = V_i$. Ainsi, nous avons pris le soin de supprimer les colonnes des films n'ayant pas été notés mais aussi les colonnes qui se retrouvaient vides après suppression de 20% des données. Ces suppressions ont été réalisées par la fonction set_train_test. Cela révèle l'une des faiblesses de cet algorithme à savoir qu'il ne peut pas proposer de notes dans le cas où l'item n'a reçu aucune note ou encore lorsque l'utilisateur n'en a donné aucune. Pour les items et utilisateurs concernés, nous pouvons faire alors appel par exemple à un autre algorithme capable de leur prédire des notes. On obtiendrait alors un algorithme hybride.

Critère d'arrêt Après chaque mise à jour des matrices U et M, il faut s'assurer si le critère d'arrêt est vérifié ou non. Dans notre cas, il s'agit d'une part de calculer le RMSE à l'étape courante et de le comparer avec celui de l'étape précédente puis vérifier s'il est plus petit qu'un certain seuil (ici 0.0001). L'autre critère d'arrêt consiste à limiter le nombre d'itérations. En effet, si le temps de convergence est trop lent et que la valeur du RMSE ne varie presque plus, il semble naturel de limiter les itérations afin d'optimiser le temps d'exécution. Ici, 20 itérations semblent déjà donner des résultats satisfaisants.

Choix des paramètres λ et K Les paramètres λ et K doivent être choisis de sorte à minimiser le RMSE. On réalise ainsi une boucle respectivement pour plusieurs valeurs de λ puis de K. Les paramètres minimisant le RMSE pour chaque base de données ont été choisis à l'aide des

graphiques suivants :

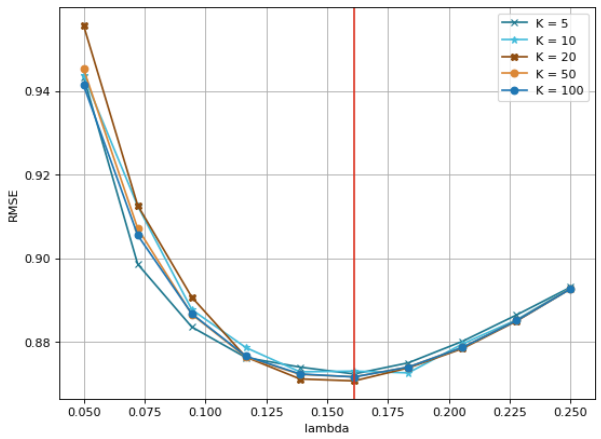


FIGURE 11 – RMSE en fonction du paramètre de régularisation - Movilens

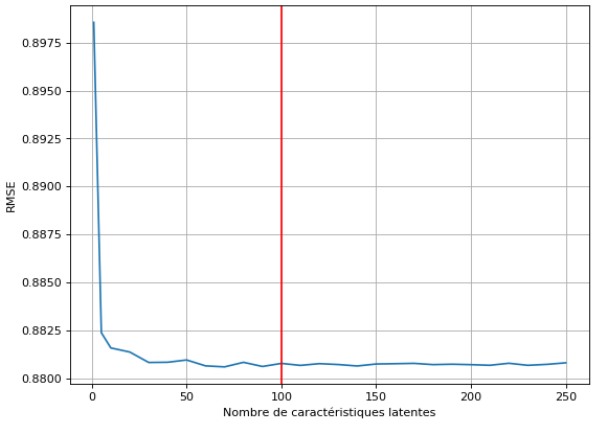


FIGURE 12 – RMSE en fonction de K - Movilens

Sur le graphe de gauche, nous pouvons constater une tendance globale pour différentes valeurs de K : le minimum du RMSE est obtenu plusieurs fois pour $\lambda = 0.16$. Concernant le graphe de droite, on observe qu'à partir de 100 caractéristique latentes il y a très peu de variation pour le RMSE et il reste également très bas. On réalise le même raisonnement pour les bases de données *Jester* à l'aide des graphes suivants :

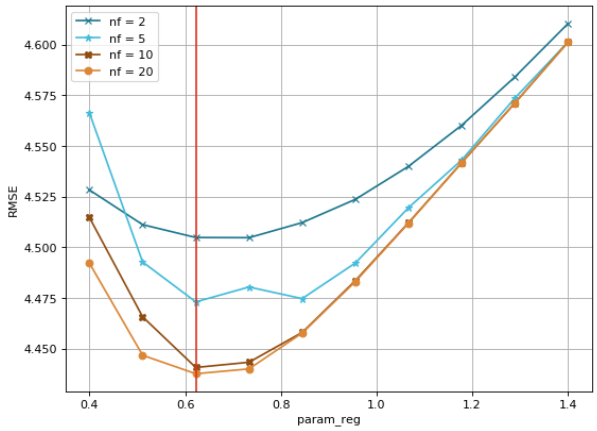


FIGURE 13 – RMSE en fonction du paramètre de régularisation - Jester4

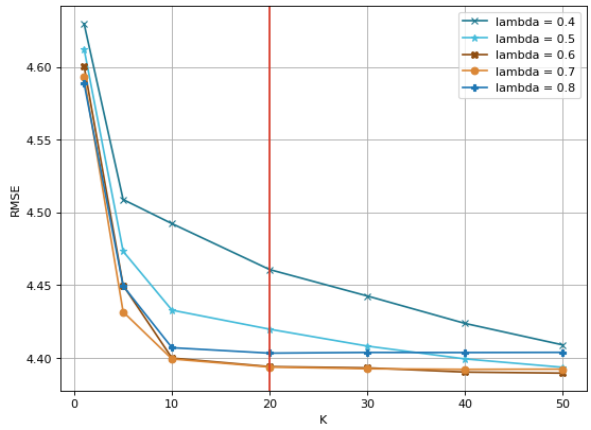


FIGURE 14 – RMSE en fonction de K - Jester4

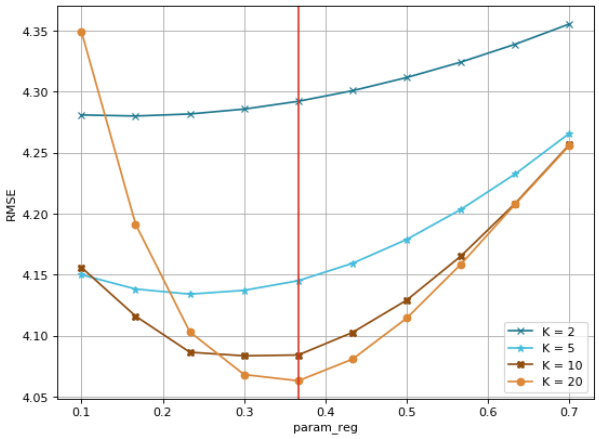


FIGURE 15 – RMSE en fonction du paramètre de régularisation - Jester12

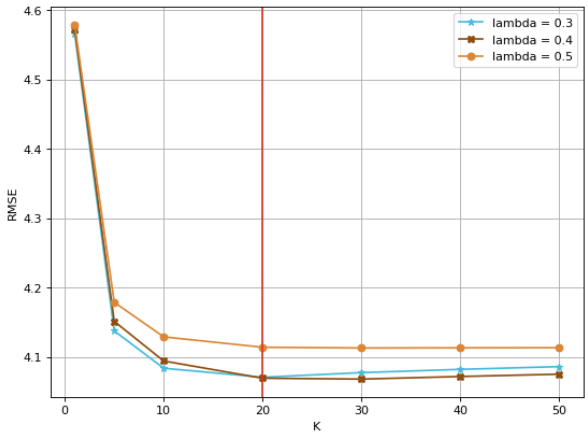


FIGURE 16 – RMSE en fonction de K - Jester12

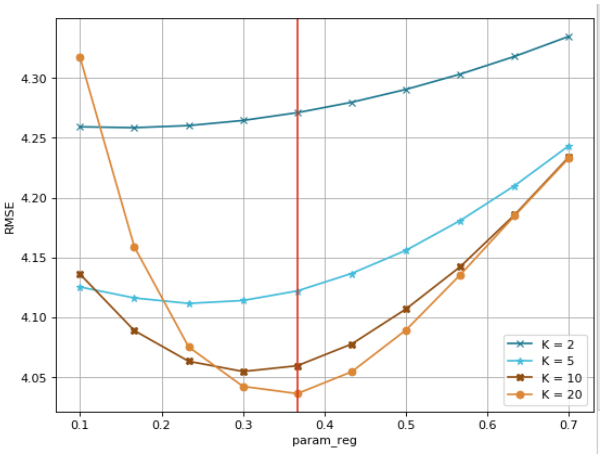


FIGURE 17 – RMSE en fonction du paramètre de régularisation - Jester11

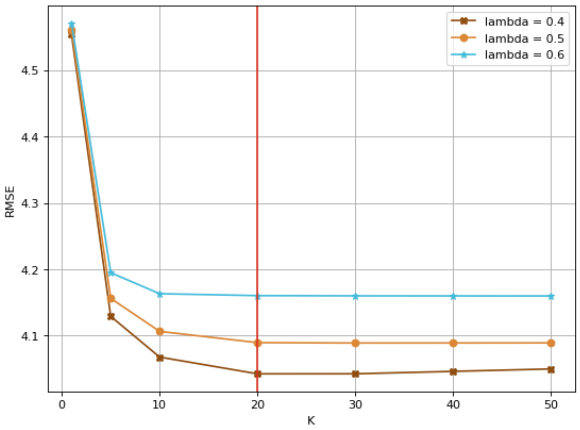


FIGURE 18 – RMSE en fonction de K - Jester11

Ces paramètres sont résumés dans le tableau suivant :

Jeu de données	λ	K
MovieLens	0.16	100
Jester Dataset4	0.62	20
Jester Dataset12	0.36	20
Jester Dataset11	0.37	20

Performance de l'algorithme de moindres carrés alternés Nous avons représenté les calculs de MAE et de RMSE pour chaque jeu de données en ayant choisi les paramètres optimaux.

Jeu de données	MAE	RMSE
MovieLens	0.6830	0.8660
Jester Dataset4	3.276	4.393
Jester Dataset12	3.190	4.062
Jester Dataset11	3.191	4.035

On constate que pour le jeu de données Movilens, nous obtenons à peu près les mêmes résultats que précédemment. Cependant, pour les différentes bases de données Jester nous obtenons de meilleurs résultats que pour les algorithmes précédents. Cela peut s'expliquer par le fait que les bases de données Jester sont plus importantes et donc l'algorithme de moindres carrés alternés est plus performant comme nous l'avions dit en introduction de cette partie.

VIII Conclusion

A travers ce projet nous avons développé quatre algorithmes de recommandation à filtrage collaboratif. Le prédicteur de base est l'algorithme dont l'implémentation est la plus simple. Il fournit un seuil de référence aux autres algorithmes de recommandation pour la performance, qui indique si un algorithme est adapté ou non à un jeu de données. La performance de cet algorithme est donc relativement faible, mais il est robuste car il donne une prédiction dans tous les cas. Il peut donc être utilisé par d'autres algorithmes lorsque ceux-ci ne fonctionnent pas dans certains cas extrêmes. Ensuite, l'algorithme des plus proches voisins est également un algorithme basé sur la mémoire. Il fournit une meilleure prédiction que le prédicteur de base grâce à un calcul de similarité entre les utilisateurs qui permet de déterminer ses plus proches voisins. Cependant il comporte de nombreuses failles, par exemple dans le cas de l'arrivée d'un nouvel utilisateur dans le jeu de données. D'autre part, l'algorithme de descente de gradient stochastique (Funk SVD) et l'algorithme des moindres carrés alternés sont tous deux des algorithmes de factorisation matricielle. Ils sont plus efficaces sur de grandes bases de données. En effet, à ce stade, l'espace de stockage devient insuffisant pour les algorithmes basés sur la mémoire. D'ailleurs, il n'a pas été possible de faire tourner l'algorithme des plus proches voisins sur les deux plus gros jeux de jester car ils contiennent trop d'utilisateurs. Sur les deux plus petits jeux de données, l'algorithme de Funk SVD donne de moins bons résulats que le prédicteur de base et l'algorithme des plus proches voisins. Toutefois, la performance s'améliore si l'on supprime les utilisateurs ou les items qui n'ont pas de notes, il s'agit donc d'une faille de l'algorithme. Enfin, l'algorithme des moindres carrés alternés a fourni les meilleurs résultats sur tous les jeux de données Jester. Cependant, il ne permet pas du tout de donner de prédiction pour les items et utilisateurs n'ayant pas de notes car cela engendre des problèmes de matrices singulières. Ces quatre algorithmes donnent un aperçu de l'approche par filtrage collaboratif utilisé par de nombreuses plateformes.

ANNEXE A

ALGORITHMES

FONCTIONS TRANSVERSALES

```
######################## IMPORTATION DES DONNEES MOVIELENS ############################
########################      Fichier python : data.py     ############################
import numpy as np
import random

#variables globales
nb_movies = 9742
nb_users = 610
movie_dict = dict() #va contenir le movieID en clé et son indice en valeur
R = np.zeros((nb_users,nb_movies),dtype=float) #matrice contenant les ratings

def create_movie_dict():
    try:
        with open("movies.csv", 'r') as f:
            index = 0
            for line in f.readlines()[1:] :
                vec = line.split(',')
                movie_dict[int(vec[0])] = index
                index += 1
        return movie_dict
    except FileNotFoundError:
        print("fichier manquant")
    except PermissionError:
        print("permission non accordée")

#fonction pour créer la matrice R
def csv_to_matrix() :
    global R
    try:
        with open("ratings.csv", 'r') as f:
            for line in f.readlines()[1:] :
                vec = line.split(',')
```

```
                u=int(vec[0])-1 #réindexe les indices des utilisateurs à partir de 0
                i=int(vec[1])
                R[u][movie_dict[i]]=float(vec[2])
            return R
    except FileNotFoundError:
        print("fichier manquant")
    except PermissionError:
        print("permission non accordée")

#la fonction set_train_test :
    #-génère les indices des couples user-item pour lesquels on a une note
    #-sélectionne aléatoirement 20% des indices pour constituer le set de test
def set_train_test() :
    nb_ratings = len(np.where(R!=0)[0])
    size_test_set = int(0.2*nb_ratings)

    R_train = np.copy(R)
    matUI = np.zeros(np.shape(R_train))
    ind_users,ind_items = np.where(R_train>0)
    n = len(ind_users)

    for k in range(size_test_set) :
        x = random.randint(0,n-1) #choisit un indice au hasard
        u = ind_users[x] #utilisateur sélectionné par cet indice
        i = ind_items[x] #item sélectionné par cet indice
        matUI[u,i] = R_train[u,i]
        R_train[u,i] = 0
        ind_users = np.delete(ind_users,x)
        ind_items = np.delete(ind_items,x)
        n -= 1

    list_U,list_I = np.where(matUI != 0)
    test_set = matUI[list_U,list_I]

    # Les lignes commentées ci-dessous permettent la suppression des colonnes
    # et lignes vides de la matrice R_train. Il est nécessaire de les décommenter
    # pour faire tourner l'algorithme des moindres carrés alternés (ALS)
    """
    x = np.where(~R_train.any(axis=1))[0]
    y = np.where(~R_train.any(axis=0))[0]
    R_train = np.delete(R_train,x,axis=0) #supprime lignes vides
    R_train = np.delete(R_train,y,axis=1) #supprime colonnes vides
    matUI = np.delete(matUI,x,axis=0) #supprime lignes vides
    matUI = np.delete(matUI,y,axis=1) #supprime colonnes vides
    list_U,list_I = np.where(matUI != 0)
    test_set = matUI[list_U,list_I]
    """
    nb_ratings = len(np.where(R_train !=0)[0])
    size_test_set = len(list_U)
```

```
    return R_train, test_set, list_U, list_I, size_test_set, nb_ratings

#test du programme :
create_movie_dict()
csv_to_matrix()
R_train, test_set, list_U, list_I, size_test_set, nb_ratings = set_train_test()

######################## IMPORTATION DES DONNEES JESTER ##########################
###################    Fichier python : data_jokes.py  ##########################
import pandas as pd
import numpy as np
import random

#*********************************************************
# cration de la matrice des notations R
#*********************************************************
# Le fichier excel doit être dans le même dossier que le script,
# sinon utiliser le chemin absolu.
df = pd.read_excel (r'[final] April 2015 to Nov 30 2019
- Transformed Jester Data - .xlsx', header=None, index_col=None)
R = df.to_numpy()
# La première colonne de R correspond au nombre de blagues évaluées
# par chaque utilisateur
nb_rating_jokes = R[:,0]
# élimination de la première colonne de R
R = np.delete(R,0,1)
# variables globales
nb_users,nb_jokes = np.shape(R)

#*********************************************************
# fonction set_train_test :
#*********************************************************
def set_train_test():
    """
   Cette fonction :
        - génère les indices des couples user-item pour lesquels on a une note
        - sélectionne aléatoirement 20% des indices pour constituer le set de test
    renvoie :
        - R_train  : matrice des notations R pour laquelle on a supprimé
        aléatoirement 20 pourcent des notes.
        - test_set : liste des notes supprimées de R (ie valeurs manquantes de R_train)
        - list_U   : liste des indices des utilisateurs supprimés de R
        - list_I   : liste des indices des items supprimés de R
    """

    nb_ratings = int(sum(nb_rating_jokes))
    size_test_set = int(0.2*nb_ratings) #20 pourcent du nb de ratings

    R_train = np.copy(R)
    matUI = 99*np.ones(np.shape(R_train))
```

```
    ind_users,ind_items = np.where(R_train != 99)
    n = len(ind_users)

    for k in range(size_test_set) :
        x = random.randint(0,n-1) #choisit un indice au hasard
        u = ind_users[x] #utilisateur sélectionné par cet indice
        i = ind_items[x] #item sélectionné par cet indice
        matUI[u,i] = R_train[u,i]
        R_train[u,i] = 99 #supprime la note dans R_train
        ind_users = np.delete(ind_users,x)
        ind_items = np.delete(ind_items,x)
        n -= 1

    list_U,list_I = np.where(matUI != 99)
    test_set = matUI[list_U,list_I]

    # Les lignes commentées ci-dessous permettent la suppression des colonnes
    # et lignes vides de la matrice R_train. Il est nécessaire de les décommenter
    # pour faire tourner l'algorithme des moindres carrés alternés (ALS)
    """
    x = np.where(~(R_train-99).any(axis=1))[0]
    y = np.where(~(R_train-99).any(axis=0))[0]
    R_train = np.delete(R_train,x,axis=0) #supprime lignes vides
    R_train = np.delete(R_train,y,axis=1) #supprime colonnes vides
    matUI = np.delete(matUI,x,axis=0) #supprime lignes vides
    matUI = np.delete(matUI,y,axis=1) #supprime colonnes vides
    list_U,list_I = np.where(matUI != 99)
    test_set = matUI[list_U,list_I]
    """
    nb_ratings = len(np.where(R_train !=99)[0])
    size_test_set = len(list_U)
    return R_train, test_set, list_U, list_I, size_test_set, nb_ratings

#*********************************************************
# test du programme :
#*********************************************************
R_train, test_set, list_U, list_I, size_test_set,
nb_ratings = set_train_test()
```

```python
############### Test de l'efficacité d'un algorithme - MAE et RMSE ##############
###############      Fichier python : efficacite_test.py          ##############
import numpy as np

# Les 2 fonctions suivantes prennent en argument une liste contenant les notes réelles
# et une liste contenant les prédictions réalisées par l'algorithme

#calcule le coefficient MAE
def MAE(notes, predic) :
    somme = 0
    for x in range(len(predic)) :
        somme += abs(predic[x]-notes[x])
    return somme/len(predic)

#calcule le coefficient RMSE
def RMSE(notes, predic) :
    somme = 0
    for x in range(len(predic)) :
        somme += ((predic[x]-notes[x])**2)
    return np.sqrt(somme/len(predic))


# Pour utiliser ces fonctions, il faut :
#- Mettre les prédictions obtenues dans une liste
#- Récupérer la liste test_set renvoyée par la fonction set_train_test
    #=> c'est la liste des notes supprimées
#- Appeler efficacite_test.MAE(...) sur ces listes
```

BASELINE PREDICTOR

```
############################ PREDICTEUR DE BASE ##############################
################ Fichier python : baseline_predictor.py ######################

import numpy as np
import data
import efficacite_test

#chargement des données
R_train, test_set, list_U, list_I, size_test_set, nb_ratings = data.set_train_test()
nb_users, nb_movies = np.shape(R_train)
m = np.sum(R_train)/nb_ratings #moyenne
list_bi=list()
list_bu=list()


def compute_biais(B) : #B est le terme de correction
    global list_bi
    global list_bu
    #biais des movies :
    for i in range(nb_movies) :
        Ui = np.where(R_train[:,i]>0)[0]
        sum_i = np.sum(R_train[Ui,i])
        if len(Ui) == 0:
            list_bi.append(0)
        else :
            list_bi.append((sum_i-len(Ui)*m)/(len(Ui)+B))

    #biais des users :
    for u in range(nb_users) :
        somme = 0
        n = 0
        for i in range(nb_movies) :
            if R_train[u,i] != 0 :
                somme += (R_train[u,i]-list_bi[i]-m)
                n += 1
        if n == 0 :
            list_bu.append(0)
        else :
            list_bu.append(somme/(n+B))
    return list_bi, list_bu

def score(u,i) :
    return (m+list_bi[i]+list_bu[u])

def test_score() :
    compute_biais(20)
    predic = list()
    for x in range(len(list_U)) :
```

```
        score_x = score(list_U[x],list_I[x])
        if score_x > 5 :
            score_x = 5
        if score_x < 0.5 :
            score_x = 0.5
        predic.append(score_x)
    print("scores obtenus par le baseline predictor : ")
    print("MAE = ", efficacite_test.MAE(test_set, predic))
    print("RMSE = ", efficacite_test.RMSE(test_set, predic))

test_score()
```

NEAREST NEIGHBORS

```
#################### ALGORITHME DES PLUS PROCHES VOISINS ######################
#################     Fichier python nearest_neighbors.py     #################
import numpy as np
import data
import baseline_predictor
import efficacite_test

#chargement des données
R_train, test_set, list_U, list_I, size_test_set, nb_ratings = data.set_train_test()
nb_users, nb_movies = np.shape(R_train)
m = np.sum(R_train)/nb_ratings #moyenne

#matrice pour stocker les similarités :
S = np.triu(np.zeros((nb_users,nb_users),dtype=float),k=1)
moy = list() #va contenir les moyennes de utillisateurs
ecart = list() #va contenir sqrt(somme(R[u,i]-mu)^2)
T = 50 #nb requis de films communs

#initialisation matrice des voisins
N = list()

#données du baseline predictor => on y fait appel quand l'algorithme des plus
#proches voisins échoue
list_bi=list()
list_bu=list()
baseline_predictor.compute_biais(20)


#calcule la moyenne et variance des notes de chaque
#utilisateur et les stocke dans une liste
def list_moy_ecart() :
    for u in range(nb_users) :
        somme = 0
        notes = np.where(R_train[u,:]>0)[0] #indices des items que u a noté
        mu = np.mean(R_train[u,notes]) #moyenne des notes données par l'utilisateur u
        moy.append(mu)

        for i in range(nb_movies) :
            if R_train[u,i] != 0 :
                somme += (R_train[u,i]-mu)**2
        ecart.append(np.sqrt(somme))

# Calcule la similarité entre deux utilisateurs
def similarity(u,v) :
    #u1 et v1 sont les sets des indices des films que les utilisateurs u et v
    #ont noté
    #la longueur de l'intersection de deux sets = nb de films en communs
    u1 = set(np.where(R_train[u,:] >0)[0])
```

```
    v1 = set(np.where(R_train[v,:] >0)[0])
    intersection = list(u1 & v1)
    # 1) si l'intersection est vide => sim = 0
    # 2) si un utilisateur a toujours donné la même note à tous les films
    #    (écart[u] =0) => l'algortihme échoue : sim = 0
    if len(intersection)<T or ecart[u]==0 or ecart[v]==0:
        sim = 0
    else :
        somme = 0
        for i in intersection : #si i fait parti des items notés par u et v
            somme += (R_train[u,i] - moy[u])*(R_train[v,i] - moy[v])
        sim = somme/(ecart[u]*ecart[v])
    return sim

# Calcule une matrice triangulaire supérieure des similarités entre tous
# les utilisateurs
def sim_matrix() :
    for u in range(nb_users-1) :
        for v in range(u+1,nb_users) :
            S[u,v]=similarity(u,v)


# Renvoie une liste des voisins de u
def neighbors(u) :
    Nu = list()
    #indices des voisins les plus similaires classés par ordre décroissant :
    ind = np.argsort(S[u,:])[::-1]
    k = 0
    while S[u,ind[k]]>0 :
        Nu.append(ind[k])
        k += 1
    return Nu

# calcule la matrice des voisins de tous les utilisateurs
def neighbors_matrix() :
    global N
    N=[neighbors(u) for u in range(nb_users)]
    return N

#calcule la prédiction de la note donné par l'utilisateur u à l'item i
def score(u,i) :
    Nui = list() # va contenir les plus proches voisins qui ont noté l'item i
    taille_Nui_max = 23 # paramètre déterminé pour optimiser la prédiction
    k = 0
    j = 0
    while j < taille_Nui_max and k < len(N[u]) :
        v = N[u][k]
        if R_train[v,i]>0: #si v a noté l'item i c'est un plus proche voisin
            Nui.append(v)
            j+=1
```

```
            k += 1
        somme1 = 0
        somme2 = 0
        for v in Nui :
            somme1 += S[u,v]*(R_train[v,i]-moy[v])
            somme2 += abs(S[u,v])
        if len(Nui) == 0: #appel au prédicteur de base si pas de voisins
            return baseline_predictor.score(u,i)
        else :
            return (moy[u]+(somme1/somme2))


def test_score() :
    global S
    list_moy_ecart()
    sim_matrix()
    S = S + S.T #rend S symétrique pour pouvoir accéder à ses indices
                #de manière équivalente
    neighbors_matrix() #calcule N

    predic = list()
    for x in range(len(list_U)) :
        score_x = score(list_U[x],list_I[x])
        if score_x > 5 :
            score_x = 5
        if score_x < 0.5 :
            score_x = 0.5
        predic.append(score_x)
    print("scores obtenus par le KNN : ")
    print("MAE = ", efficacite_test.MAE(test_set, predic))
    print("RMSE = ", efficacite_test.RMSE(test_set, predic))

baseline_predictor.test_score()
test_score()
```

DESCENTE DE GRADIENT STOCHASTIQUE

```
########################   ALGORITHME DE FUNK-SVD   ############################
##################  Fichier python : Funk_SVD_movielens.py ####################
import numpy as np

def compute_Funk_SVD (R, K, alpha, beta, nb_epochs):
    """
    Algorithme de descente de gradient stochatique pour prédire les
    entrées vides dans une matrice.

    Arguments:
        - R (array)      : tableau des utilisateurs(lignes) - items(colonnes)
        - K (integer)    : nombre de caractéristiques latentes
        - alpha (float) : taux d'apprentissage
        - beta (float)  : paramètre de régularisation
        - nb_epochs      : nombre d'itérations de descente de gradient

    Ne retourne rien. Crée et itère sur les matrices P et Q de sorte
    que R ~= P.tranpose(Q) = ^R
    """
    num_users, num_items = R.shape

    # initialisation des matrices P et Q avec des valeurs aléatoires issues
    # d'une loi gaussienne centrée et de variance 1/K
    P = np.random.normal(scale = 1./K, size=((num_users),K))
    Q = np.random.normal(scale = 1./K, size=((num_items),K))

    # initialisation des biais
    b_u = np.zeros(num_users) #vecteur biais des utilisateurs
    b_i = np.zeros(num_items) #vecteur biais des items
    b = np.mean(R[np.where(R != 0)]) #moyenne des notes différentes de 0
                                     #(entrées non vides de R)

    # crée la liste des samples d'entraînement de l'algorithme (liste de tuples)
    samples =  [
            (i,j,R[i,j])
            for i in range(num_users)
            for j in range (num_items)
            if R[i,j] != 0
            ]

    def get_rating(i,j):
        """
        Obtient la note prédite de l'utilisateur i et de l'item j : ^rij
        """
        prediction = b + b_u[i] + b_i[j] + P[i,:].dot(Q[j,:].T)
        return prediction

    def one_step_stochastic():
```

```
    """
    fonction qui calcule une seule itération de gradient stochastique
    """
    for i,j,r in samples:

        # calcule la prediction er l'erreur
        prediction = get_rating(i,j) # rij
        e = (r - prediction) # e = rij - ^rij

        # mise à jour des biais
        b_u[i] += alpha * (e - beta * b_u[i])
        b_i[j] += alpha * (e - beta * b_i[j])

        # mise à jour des matrices P et Q
        P[i,:] += alpha *(e * Q[j,:] - beta * P[i,:])
        Q[j,:] += alpha *(e * P[i,:] - beta * Q[j,:])

def full_matrix():
    """
    Calcule la matrice ^R utilisant les biais, P et Q
    """
    return b + b_u[:,np.newaxis] + b_i[np.newaxis:,] + P.dot(Q.T)

def total_error():
    """
    Calcule l'erreur totale
    """
    indice_lig, indice_col = np.where(R != 0)
    predicted = full_matrix()
    error = 0
    for i,j in zip (indice_lig, indice_col):
        error = error + pow(R[i,j] - predicted[i,j], 2)
    return np.sqrt(error)

# performe la descente de gradient stochastique pour nb_epochs itérations
#-------------------------------------------------------------------------------
training_process = []
for i in range(nb_epochs):
    np.random.shuffle(samples) #mélange le dataset :
                               #réalise le côté "stochastique"
    one_step_stochastic() # une itération de descente gradient
    tot_error = total_error()
    training_process.append((i,tot_error)) #utile pour tracer l'erreur totale
                                           #en fonction de l'epoch i
    R_predicted = full_matrix()
return R_predicted,training_process
```

```
######## Calcul des paramètres optimaux & performance pour la Funk-SVD  ############
##################  Fichier python : Run_Funk_SVD.py ##############################

### importation des librairies nécessaires
#################################################
import numpy as np
import numpy.linalg as npl
import matplotlib.pyplot as plt
import pandas as pd

import data
import Funk_SVD_movielens as Funk_SVD
import efficacite_test

#################################################
# Récupération et mise en forme des données
#################################################
data.create_movie_dict() # retourne le dictionnaire movie_dict
data.csv_to_matrix() # charge les données et les met dans la matrice R
R_train, test_set, list_U, list_I, size_test_set, nb_ratings = data.set_train_test()

#################################################
# Détermination de K
#################################################
# on choisit K comme étant le nombre de valeurs singulières dominantes de R_train
#------------------------------------------------
u, s, vh  = npl.svd(R_train,full_matrices=False) #SVD de rang réduit
x = np.arange(1,min(np.shape(R_train))+1)

fig = plt.figure(0)
plt.figure(figsize=((16,4)))
plt.plot(x,s)
plt.title(u"Valeurs singulières de la matrice des notations")
plt.xlabel("composantes")
plt.ylabel("valeur singulière")
plt.xticks(np.arange(0,min(np.shape(R_train))+1, 20))
plt.grid()
plt.savefig('Movielens_singularValue.png', bbox_inches='tight', dpi = 300,
format = 'png')
plt.show #à mettre après savefig

#################################################
# Calcul de la Funk_SVD
#################################################
R_predicted, training_process = Funk_SVD.compute_Funk_SVD(R_train, K=200,
alpha=0.095, beta=0.02, nb_epochs=10)

#################################################
# Tracé de l'erreur à chaque epoch pour vérifier qu'elle diminue
```

```
# et donc que l'algorithme converge bien.
#*******************************************
x = [x for x, y in training_process]
y = [y for x, y in training_process]
plt.plot(x, y)
plt.xticks(x, x)
plt.xlabel("Iterations")
plt.ylabel("Erreur totale")
plt.grid(axis="y")

#*******************************************
# Détermination du meilleur learning rate (alpha)
#*******************************************
# Nous faisons tourner notre programme avec différentes valeurs de alpha
# et traçons l'erreur à chaque itération.
# Le but est de trouver le alpha donnant la plus petite erreur.
nb_epochs = 10
learning_rates = [0.09,0.095,0.01,0.015,0.020,0.025]
all_training_process = np.zeros((nb_epochs,len(learning_rates)))
x = np.arange(0,nb_epochs)

for i in range(0,len(learning_rates)):
    R_predicted2,training_process2 = Funk_SVD.compute_Funk_SVD(R_train, K=200,
    alpha=learning_rates[i], beta=0.02, nb_epochs=nb_epochs)
    all_training_process[:,i] = [y for x, y in training_process2]

plt.figure(1)
plt.figure(figsize=((10,4)))
ax = plt.subplot(111)
for i in range(0,len(learning_rates)):
    ax.plot(x,all_training_process[:,i], label = str(learning_rates[i]))

# Mise en forme du graphique
box = ax.get_position()
ax.set_position([box.x0, box.y0, box.width * 0.8, box.height])
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.xlabel("Iterations")
plt.ylabel("Erreur totale")
plt.xticks(x, x)
plt.savefig('Movielens_bestLearningRate.png', bbox_inches='tight', dpi = 300,
format = 'png')
plt.show()

# exportation de all_training_processes dans un fichier excel
# pour mieux visualiser les résultats
columns = list(map(str,np.round(learning_rates, decimals=4)))
df = pd.DataFrame( all_training_process,columns=columns)
df.to_excel("Movielens_all_training_process.xlsx")

#*******************************************
```

```
# Calcul de l'efficacité de l'algorithme
#*****************************************
list_predic = R_predicted[list_U,list_I] # On met nos prédictions dans une liste
print()
print ("erreur MAE de l'algorithme Funk_SVD: ")
print(efficacite_test.MAE(test_set,list_predic))
print()
print ("erreur RMSE de l'algorithme Funk_SVD: ")
print(efficacite_test.RMSE(test_set,list_predic))
```

MOINDRES CARRES ALTERNES

```
################## ALGORITHME DES MOINDRES CARRES ALTERNES ####################

def Init_M(R_train, K) :

    m,n = np.shape(R_train)
    M=np.random.rand(K, n)
    #On remplit la 1ere ligne de M avec les moyennes des films
    for i in range(n) :
        ind = np.where(R_train[:,i]!=0)[0]
        moyenne = np.mean(R_train[ind,i])
        M[0,i] = moyenne

    return M

def Alternating_Least_Square(R_train, param_reg, K, it_max) :

    """
    Paramètres :

        - R_train : User/Item matrice

        - param_reg : Terme de régularisation pour les items/users.
        Permet d'éviter l'overfitting

        - K : Nombre de facteurs latents

    """
    m,n = np.shape(R_train)

    U=np.zeros((K, m), dtype=float)
    start_time = time.time()

    #ETAPE 1 : Initialisation de la matrice M
    M= Init_M(R_train, K)
    it = 0 #nb d'iterations
    E = np.eye(K)
    cond0 = 10 #Contiendra le RMSE de l'itération courante
    cond1 = cond0+1 #Contiendra le RMSE de l'itération précédente

    while (it<it_max) and (np.abs(cond0-cond1)> 0.0001) :

        it=it+1

        #ETAPE 2 : On fixe M et on met à jour U
        for u in range(0, m) : #Parcours de ch colonne de U
            ind1 = np.where(R_train[u,:]>0)[0] #indices des items que u a notés
            m_u = len(ind1) #représente le nb d'items notés par u
            M_Iu = M[:,ind1] #Selection des vecteurs latents m_i
```

```
                        #representant les items i que u a notés
            R_u_Iu = R_train[u, ind1] #représente toutes les notations de u
                                      #(taille = m_u)
            V_u= np.dot(M_Iu, np.transpose(R_u_Iu))
            A_u= np.dot(M_Iu, np.transpose(M_Iu)) + param_reg*m_u*E
            U[:,u]= npl.solve(A_u, V_u)

        #ETAPE 3 : On fixe U et on met à jour M
        for i in range(0, n) :
            ind2 = np.where(R_train[:,i]>0)[0] #représente les indices
                                               # des utilisateurs ayant noté i
            n_i = len(ind2) #représente le nb de notes pour l'item i
            U_Ii = U[:,ind2]
            R_Ii_i = R_train[ind2, i]
            V_i = np.dot(U_Ii, R_Ii_i)
            A_i = np.dot(U_Ii, np.transpose(U_Ii)) + param_reg*n_i*E
            M[:,i]= npl.solve(A_i, V_i)

        cond1 = cond0
        #On met à jour les prédictions pour le calcul du RMSE
        predic = list()
        for x in range(len(list_U)) :
            predic.append(np.dot(np.transpose(U[:, list_U[x]]), M[:,list_I[x]]))
            #la prediction correspond au produit scalaire entre les Uu et Mi
        cond0 = RMSE(test_set, predic) ;
        mae = MAE(test_set, predic) ;

    return U, M, cond0, mae, start_time
```