UNIVERSITÉ DU QUÉBEC À MONTRÉAL

ÉVALUATION DES ALGORITHMES DE RECOMMANDATION VIA DES ORACLES

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

MOHAMED ALI MAMI

NOVEMBRE 2024

REMERCIEMENTS

Je tiens à remercier Monsieur Florent Avellaneda, professeur au département d'informatique de l'Université du Québec à Montréal, pour son soutien et sa guidance tout au long de la réalisation de ce mémoire. Sa disponibilité, son encadrement et ses conseils, ont grandement contribué à enrichir ma recherche et approfondir mes connaissances.

Je souhaite également remercier mes parents pour leur soutien et leur encouragement tout au long de mon parcours académique. Leur confiance et leur soutien ont été une source constante de motivation.

Enfin, je remercie l'Université du Québec à Montréal pour m'avoir offert l'opportunité de réaliser cette maîtrise. L'institution a été un cadre très enrichissant et j'ai beaucoup appris grâce aux ressources et aux opportunités offertes.

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

ACRONYMES

KNN K-Nearest Neighbors.

SVD Singular Value Decomposition.

NMF Non-negative Matrix Factorization.

B-BMF Bayesian-Boolean Matrix Factorization.

ML MovieLens.

RT Random Trees.

LLM Large Language Model.

GPT Generative Pre-trained Transformer.

O-NMF Oracle-Non-negative Matrix Factorization.

O-ADA Oracle-Arbres de Décision Aléatoire.

D Default.

B1 Bagging.

B2 BaggingV2.

RÉSUMÉ

Les systèmes de recommandation sont des outils très importants dans de nombreux domaines tels que le commerce électronique (e-commerce), les médias sociaux et le divertissement en ligne. Ils aident les utilisateurs à découvrir des produits ou contenus pertinents en se basant sur leurs préférences passées ou les comportements d'autres utilisateurs ou en combinant les deux ensemble.

Comme de nombreux articles l'ont présenté et discuté, il existe une grande variété d'algorithmes de recommandation, allant de la factorisation de matrices à l'apprentissage profond. Cependant, il est souvent difficile pour les chercheur de déterminer quel algorithme est le plus performant en raison du manque de jeux de données disponibles et des différentes conditions et contraintes expérimentales. Pour la recommandation des films par exemple, les évaluations actuelles se basent principalement sur quelques ensembles de données réels qui sont accessibles au public, mais ces ensembles peuvent ne pas toujours refléter toutes les situations possibles afin de déterminer quel algorithme performe le mieux.

Pour adresser ces défis, nous proposons de générer des jeux de données synthétiques en utilisant des Oracles. Les Oracles permettent de créer des ensembles de données artificiels avec des caractéristiques contrôlées, facilitant ainsi l'évaluation des algorithmes dans des conditions variées et bien définies.

Nous proposons, également, d'explorer le concept de bagging en utilisant plusieurs algorithmes de recommandation déjà existants. Ce concept consiste à créer plusieurs modèles à partir de sous-ensembles variés de données pour améliorer la robustesse et la précision des recommandations. Ce qui permet de tirer parti des forces des différents algorithmes et de compenser leurs faiblesses respectives.

Le but de ce mémoire est donc d'effectuer une évaluation comparative des algorithmes de recommandation de la littérature en utilisant les jeux de données synthétiques générés par les Oracles et nos versions de bagging, de tester également les performances de ces algorithmes en conditions bruitées pour évaluer leur capacité à gérer le bruit et maintenir la précision des recommandations pour fournir des conclusions sur la robustesse des algorithmes et permettre de mieux comprendre leurs dynamiques de performance.

Mots clés : Algorithmes de recommandation, Évaluation des performances, Analyse comparative, Génération de données, Oracle, Bagging, Gestion du bruit.

INTRODUCTION

Les systèmes de recommandation jouent un grand rôle dans notre société, en facilitant la navigation à travers une multitude de choix disponibles dans divers domaines tels que le commerce électronique, les médias sociaux, et les plate-formes de streaming. Ces algorithmes sont conçus pour filtrer et personnaliser les informations, offrant ainsi aux utilisateurs des suggestions pertinentes basées sur leurs préférences et comportements antérieurs.

Bien que les sites web de commerce électronique, souvent appelés « e-commerce websites », soient des exemples notables d'application de ces systèmes, où ils suggèrent aux utilisateurs des produits ou des services sur la base des informations disponibles, telles que les recherches antérieures, l'âge, le sexe et d'autres préférences (Abdulla et Designs, 2017). Leur utilisation ne se limite pas à ce seul domaine. Les systèmes de recommandation sont également omniprésents dans d'autres secteurs, comme la musique, les vêtements, les films et les séries télévisées. Par exemple, sur des plate-formes de streaming telles que Netflix et Spotify, les algorithmes recommandent des films, des séries et de la musique en fonction des préférences et des habitudes de consommation des utilisateurs. Ces recommandations permettent aux utilisateurs de découvrir de nouveaux contenus qui correspondent à leurs goûts personnels, enrichissant ainsi leur expérience de divertissement.

En effet, 35% des achats effectués par les clients sur Amazon proviennent d'un système de recommandation (MacKenzie *et al.*, 2013). Ces systèmes permettent de filtrer et de personnaliser une large quantité d'informations, ce qui simplifie la prise de décision pour les utilisateurs en leur proposant des suggestions adaptées à leurs goûts et à leurs comportements passés. Grâce à leur capacité à analyser les préférences individuelles et à prédire les intérêts futurs des utilisateurs, les systèmes de recommandation améliorent l'expérience utilisateur en rendant les interactions plus pertinentes et personnalisées. En offrant des recommandations ciblées, ces systèmes contribuent non seulement à optimiser les choix des consommateurs mais également à renforcer la fidélité des utilisateurs envers les plate-formes qui les utilisent.

Les systèmes de recommandation facilitent la manière dont nous interagissons avec les diverses plate-formes, rendant nos expériences en ligne plus fluides et adaptées à nos besoins individuels. Leur capacité à évoluer et à s'adapter aux préférences changeantes des utilisateurs permet de continuer à enrichir les interactions avec les différentes plate-formes.

Contexte

Dans le domaine des recommandations de films, les systèmes de recommandation de films sont destinés à donner des suggestions aux utilisateurs sur la base des caractéristiques qu'ils aiment ou sur la base de la similitude entre eux et d'autres utilisateurs avec plus ou moins les mêmes goûts. Un système de recommandation de films très performant suggérera des films qui correspondent aux similitudes avec le plus haut degré de performance (Jayalakshmi *et al.*, 2022). Ces systèmes utilisent une variété d'algorithmes pour estimer la note qu'un utilisateur pourrait donner à un film particulier, en évaluant les informations disponibles sur les habitudes des utilisateurs et en faisant des suggestions à partir de ces informations (Alyari et Jafari Navimipour, 2018). En analysant ces données, les systèmes de recommandation peuvent identifier des patterns et des tendances qui permettent de prédire les préférences individuelles avec une certaine précision. Par exemple, certains systèmes utilisent des techniques de filtrage collaboratif pour comparer les préférences d'un utilisateur avec celles d'autres utilisateurs ayant des goûts similaires. D'autres approches, comme le filtrage basé sur le contenu, analysent les caractéristiques des films eux-mêmes (comme le genre, les acteurs principaux, le réalisateur etc...) pour recommander des films similaires à ceux déjà appréciés par l'utilisateur (Li *et al.*, 2023). Ces systèmes sont particulièrement importants dans l'industrie du divertissement, où ils aident non seulement à guider les choix des consommateurs mais aussi à promouvoir des films moins connus qui pourraient correspondre aux goûts uniques des utilisateurs. En facilitant la découverte de nouveaux contenus tout en maximisant la satisfaction de l'utilisateur, les systèmes de recommandation contribuent à enrichir l'expérience cinématographique des utilisateurs et à faire confiance aux recommandations des plate-formes de streaming et les services de divertissement en ligne.

Problématique

Bien que de nombreux algorithmes de recommandation de films existent dans la littérature, et que divers algorithmes de calcul ont été proposés et utilisés pour accroître l'efficacité des systèmes de recommandation (Cami *et al.*, 2017), il est crucial que l'évaluation des différents algorithmes de recommandation soit cohérente et rigoureuse pour permettre des comparaisons valables des résultats. Des méthodes d'évaluation inconsistantes peuvent mener à des conclusions erronées sur la performance des algorithmes. Une étude notable de Google a démontré que lorsqu'ils ont réévalué certains algorithmes de recommandation, les résultats obtenus étaient différents de ceux rapportés dans les publications précédentes (Rendle *et al.*, 2019). Ce constat met en évidence l'importance de maintenir des standards uniformes dans les méthodes d'évaluation. L'évaluation de l'efficacité de ces algorithmes n'est pas toujours une tâche facile. A titre d'exemple, de nombreuses études ont souligné les difficultés d'accès aux ensembles de données du monde réel (Pro-

valov *et al.*, 2021). Ce problème est souvent causé par les préoccupations des entreprises concernant la confidentialité des utilisateurs (Lesnikowski *et al.*, 2021). Un des problèmes ici c'est que les approches de recommandation classiques ne requièrent généralement que des données d'interaction avec l'utilisateur comme paramètres d'entrée, tandis que les approches modernes peuvent utiliser des paramètres d'entrée plus sensibles à la protection de la vie privée, tels que l'âge, le sexe, le pays d'origine et les informations relatives à l'appareil. Cela peut permettre d'améliorer encore le pouvoir prédictif de l'approche des algorithmes de recommandation (Neumann *et al.*, 2023). Mais ces types de données réels, comme ceux de Netflix (Bennett et Lanning, 2007) ou des autres grandes plate-formes de films, sont souvent difficiles à obtenir car ils contiennent des informations sensibles sur les utilisateurs et sur leurs préférences et leurs comportements. Les entreprises craignent que le partage de ces données ne compromette la confidentialité des utilisateurs et ne conduise à des violations potentielles de la vie privée. Ce qui rend encore plus difficile l'accès aux ensembles de données réels. Pour contourner ce problème de confidentialité, de nombreux articles proposent de générer des ensembles de données synthétiques dont la structure inhérente ressemble étroitement à celle de l'ensemble de données original (Hernandez *et al.*, 2022). Bien que cette approche fournisse une solution intéressante au problème de confidentialité des données, la génération de ces ensembles de données synthétiques reste dépendante de la disponibilité d'un ensemble de données original, et la structure des ensembles de données synthétiques sera contrainte par l'ensemble de données original utilisé.

Idée proposée et objectifs

Dans ce projet de mémoire, nous proposons une idée plus radicale en générant des ensembles de données synthétiques à partir de zéro. L'idée est de créer des jeux de notes en utilisant des modèles génératif, que l'on appelle oracle. Nous utiliserons cet oracle pour générer des ensembles de données d'entraînement, potentiellement bruités, pour entraîner des algorithmes de recommandations et un autre ensemble que nous utiliserons pour la partie du test/validation pour évaluer les performances de chacun des algorithmes que nous voulons tester. Un algorithme d'apprentissage sera considéré de bonne qualité si le modèle obtenu à partir de cet ensemble de données d'entraînement fournit des prédictions proches de celles de l'oracle. L'avantage de notre approche est double. Tout d'abord, elle permet la génération d'ensembles de données très variés, permettant ainsi l'étude du comportement des différents algorithmes d'apprentissage automatique en fonction de la structure des oracles utilisés. Ensuite, et pour du plus long terme, en comparant les résultats obtenus à partir de différents algorithmes d'apprentissage sur différents oracles, cela pourrait aider à identifier les structures importantes présentes dans les ensembles de données de notes. Cette approche permet donc de surmonter les limitations des ensembles de données réels, en offrant la possibilité

de contrôler et de moduler précisément les caractéristiques des données générées facilitant ainsi l'évaluation de différents algorithmes de manière fine. Cela permet d'identifier quel algorithme fonctionne le mieux dans diverses situations et contextes, offrant ainsi une évaluation plus précise de leur efficacité respective.. Par exemple, nous pouvons introduire différents niveaux de bruits, simuler diverses distributions de notes et expérimenter avec un large interval de paramètres pour observer leur impact sur la performance des algorithmes de recommandation. Notre méthode facilite également la reproductibilité des expériences. En générant des ensembles de données synthétiques via des oracles, les chercheurs peuvent reproduire et vérifier les résultats des études précédentes sans dépendre de l'accès à des ensembles de données propriétaires ou sensibles. Cela ouvre la voie à une plus grande collaboration et à l'innovation dans le domaine des systèmes de recommandation. En développant et en affinant ces oracles, nous pouvons améliorer notre compréhension des dynamiques sous-jacentes des algorithmes de recommandation. L'étude comparative des algorithmes sur des ensembles de données générés par des oracles diversifiés offre une vision unique de leur capacité à généraliser et à s'adapter à différentes structures de données, ce qui est très important pour le développement de systèmes de recommandation plus robustes et efficaces.
L'objectif de ce mémoire peut donc se diviser en trois parties :

1. **Mettre en place le framework d'évaluation d'algorithmes de recommandation via l'utilisation d'oracles** : Cette étape consiste à concevoir un cadre méthodologique pour évaluer la performance des algorithmes de recommandation en utilisant des ensembles de données synthétiques générés par des oracles.
2. **Proposer et implémenter des oracles** : Dans cette phase, nous développerons des modèles génératifs (oracles) capables de créer des ensembles de données synthétiques. Ces oracles seront conçus pour simuler diverses configurations de données, incluant des variations dans la distribution des notes et des niveaux de bruit.
3. **Évaluer les différents algorithmes au travers du framework proposé** : Enfin, nous évaluerons les algorithmes de recommandation, y compris de nouveaux modèles proposés, en les testant dans le cadre du framework mis en place.

CHAPITRE 1

ÉTAT DE L'ART

1.1 Représentation des données

Tous les systèmes de recommandations des films reposent sur l'usage des informations sur les utilisateurs, sur les films et bien sûr les notes que les utilisateurs ont attribués aux films. Pour une meilleure illustration, les notes des utilisateurs sont généralement représentées sous la forme d'une matrice, où chaque ligne correspond à un utilisateur et chaque colonne à un film. Dans cette matrice, les cases contiennent les notes attribuées par chaque utilisateur à chaque film. Ce format matriciel permet de visualiser facilement les interactions utilisateur-film et sert pour de nombreuses techniques de recommandation. Les cases de la matrice montrent les notes, facilitant ainsi l'analyse des préférences des utilisateurs et le développement d'algorithmes de recommandation basés sur ces données.

	Matrix	Rocky	Aliens	Sharks	Avatar
Alice	3	1	1	3	1
Bob	1	2	4	1	3
Charlie	3	1	1	3	1
Diana	4	3	5	4	4

Figure 1.1 Matrice complète des notes des utilisateurs pour les films

Ci-dessus, une figure qui illustre un petit échantillon de données avec 4 utilisateurs et 5 films. Dans cette matrice, chaque ligne représente un utilisateur, chaque colonne représente un film et chaque case représente une note attribuée par un utilisateur à un film.

Il est fréquent de rencontrer des situations où certains films n'ont pas été regardés ou notés par certains utilisateurs. Cette absence de données entraîne des trous dans la matrice des évaluations. En fait, si l'on examine les valeurs de notre matrice, on s'aperçoit que plusieurs valeurs sont incomplètes compliquant ainsi le processus de recommandation (Zhang, 2022).
La figure suivante montre la même matrice, mais avec des valeurs manquantes. Ces cases vides représentent des notes qui n'ont pas encore été attribuées.

	Matrix	Rocky	Aliens	Sharks	Avatar
Alice	3		1		1
Bob	1		4	1	
Charlie	3	1		3	1
Diana		3		4	4

Figure 1.2 Matrice incomplète des notes des utilisateurs pour les films

Le défi principal est que les algorithmes de recommandation débutent leur processus d'apprentissage à partir de matrices similaires à celle présentée ci-dessus, où certaines valeurs sont manquante. Plusieurs techniques et algorithmes on été créés pour aborder ce problème des notes manquantes et comment remplir tous ces trous, chacun avec ses propres approches et méthodologies. Ces techniques visent à analyser les données des utilisateurs et des films pour fournir des recommandations personnalisées et pertinentes. Dans les sections suivantes, nous présentons les algorithmes que nous avons utilisé dans notre projet, ainsi que leurs fonctionnement.

1.2 k-Nearest Neighbors

Le K-Nearest Neighbors (les K plus proches voisins) (Zhang, 2016) est une une méthode utilisée dans les systèmes de recommandation pour prédire les préférences des utilisateurs en se basant sur la similarité entre les utilisateurs ou les articles. Cette méthode est efficace, mais elle peut être lente pour de grandes bases de données. Dans KNN, les utilisateurs et les films, sont représentés dans un espace de caractéris-

tiques. Chaque utilisateur u et chaque film v sont décrits par des vecteurs de caractéristiques $\mathbf{x}_u$ et $\mathbf{x}_v$ que l'algorithme utilisera pour faire la prédiction des notes à travers la similarité entre ces derniers.

Les caractéristiques utilisées dans l'algorithme KNN sont généralement extraites des interactions des utilisateurs avec le système. Pour les utilisateurs, ces caractéristiques peuvent être les évaluations de films qu'ils ont données, ou par exemple le temps passé sur chaque film ou les genres qu'ils préfèrent. Quant aux films, leurs caractéristiques incluent souvent des informations explicites comme le genre, les acteurs, le réalisateur, ou l'année de sortie.
La similarité entre deux vecteurs $\mathbf{x}_u$ et $\mathbf{x}_v$ est souvent mesurée par la distance euclidienne :

$$\text{sim}(u, v) = -\sqrt{\sum_{i=1}^{m}(x_{ui} - x_{vi})^2} \tag{1.1}$$

Bien que la distance euclidienne est largement utilisée pour le calcul de la similarité, il existe aussi une autre mesure pour calculer la similarité qui est également largement utilisée, il s'agit de la similarité cosinus :

$$\text{sim}(u, v) = \frac{\mathbf{x}_u \cdot \mathbf{x}_v}{\|\mathbf{x}_u\|\|\mathbf{x}_v\|} \tag{1.2}$$

Un autre concept est le paramètre k qui détermine le nombre de voisins qui seront choisis pour l'algorithme kNN. Le choix approprié de k a un impact significatif sur les performances diagnostiques de l'algorithme kNN. Un k élevé réduit l'impact de la variance causée par l'erreur aléatoire, mais fait courir le risque d'ignorer un modèle petit mais important. La clé pour choisir une valeur k appropriée est de trouver un équilibre entre le sur-ajustement et le sous-ajustement (Zhang, 2016). Pour Trouver les plus proches voisins d'un utilisateur u, on recherche les k utilisateurs les plus similaires $N(u)$ en termes de distance ou de similarité, qui peut être la distance euclidienne ou la distance de cosinus, que nous pouvons définir comme suit :

$$N(u) = \text{Top-}k(\text{sim}(u, v)) \tag{1.3}$$

Enfin, la prédiction de la note de l'utilisateur u pour le film i est pondérée par la similarité et calculée en

fonction des notes des k voisins les plus proches :

$$\hat{r}_{ui} = \frac{\sum_{v \in N(u)} \text{sim}(u, v) \cdot r_{vi}}{\sum_{v \in N(u)} \text{sim}(u, v)} \tag{1.4}$$

où r_{vi} est la note donnée par l'utilisateur v à l'article i.

1.3 BaselineOnly

L'algorithme BaselineOnly de la bibliothèque Surprise est un modèle de base utilisé aussi dans les systèmes de recommandation pour fournir des prédictions de notation basées sur des valeurs de référence simples, sans faire appel à des techniques de factorisation ou d'apprentissage avancées. Cet algorithme calcule les prédictions en tenant compte des effets globaux, des utilisateurs et des articles (films). L'algorithme BaselineOnly repose sur l'idée que les notes attribuées par les utilisateurs peuvent être décomposées en plusieurs composants :

- μ : La moyenne globale des notes dans la base de données.
- b_u : Le biais de l'utilisateur, qui représente la tendance d'un utilisateur à noter plus haut ou plus bas que la moyenne globale.
- b_i : Le biais de l'article (film), qui représente la tendance d'un film à recevoir des notes plus hautes ou plus basses que la moyenne globale.

Le modèle de prédiction de BaselineOnly est donc :

$$\hat{r}_{ui} = \mu + b_u + b_i \tag{1.5}$$

Pour calculer les biais b_u et b_i, l'algorithme minimise l'erreur quadratique moyenne des prédictions par rapport aux notes réelles, en utilisant une technique de régularisation pour éviter le sur-apprentissage. Cela peut être formulé comme une optimisation :

$$\min_{b_u, b_i} \sum_{(u,i) \in \text{trainset}} (r_{ui} - \mu - b_u - b_i)^2 + \lambda(b_u^2 + b_i^2) \tag{1.6}$$

où r_{ui} est la note réelle de l'utilisateur u pour l'article i, et λ est un paramètre de régularisation qui contrôle la complexité du modèle (Hug, 2020a).

1.4 Factorisation de Matrice

L'une des approches les plus connues dans les systèmes de recommandations, est la factorisation matricielle, qui est une technique puissante utilisée pour décomposer une matrice en produits de matrices plus simples. Cette méthode permet de capturer des relations latentes dans les données et de réduire la dimensionnalité tout en préservant les caractéristiques essentielles. Cette technique de réduction dimensionnelle est utilisée dans plusieurs domaines pour analyser les matrices utilisateur-élément. En effet, elle est largement utilisée dans les systèmes de recommandation de films. Elle décompose une matrice de notes en deux matrices de dimension inférieure, de telle sorte que la matrice originale de notes soit approximée par le produit de ces deux matrices.
Parmi les algorithmes les plus utilisés pour la factorisation de matrice, on trouve NMF (Non-negative Matrix Factorization) et SVD (Singular Value Decomposition). Nous allons maintenant explorer en détail ces approches et illustrer leurs fonctionnements à travers des exemples concrets.

1.4.1 Factorisation de Matrice Non-Négative

La Factorisation de Matrice Non-Négative (NMF) est une technique basée sur la factorisation de matrice, et elle fonctionne comme suit : Étant donné une matrice non négative X, il faut trouver les facteurs matriciels non négatifs Z et U tels que : $X \approx ZU$. Avec les contraintes $X, Z, U \in \mathbb{R}^+$, c'est-à-dire que toutes les valeurs de toutes les matrices doivent être non négatives (Green et Bailey, 2024). Cette contrainte est très importante pour l'interprétabilité des facteurs latents.
La NMF peut être appliquée à l'analyse statistique de données multivariées de la manière suivante : Étant donné un ensemble de vecteurs de données multivariées à n dimensions, les vecteurs sont placés dans les colonnes d'une matrice X de taille $n \times m$ où m est le nombre d'exemples dans l'ensemble de données. Cette matrice est ensuite approximativement factorisée en une matrice Z de taille $n \times r$ de rang inférieur résultante et une matrice U de taille $r \times m$ de rang inférieur résultante. Habituellement, r est choisi pour être plus petit que m, de sorte que Z et U sont plus petits que la matrice originale X. Comme ça, on se retrouve avec une version compressée de la matrice de données originale (Lee et Seung, 2000). Pour le cas des systèmes de recommandations, la matrice X représente la matrice de notes avec m utilisateurs et n

films. On peut maintenant facilement représenter chaque utilisateur par une ligne et chaque film par une colonne ce qui fait que la note attribuée par un utilisateur u pour un film f est tout simplement la case de la matrice $X_{u,f}$ avec u est le numéro de la ligne qui représente l'utilisateur en question et f le numéro de la colonne qui représente le film que u a noté. Puisque l'objectif de la NMF est de trouver les matrices Z et U telles que leur produit $Z \cdot U$ soit une bonne approximation de X, Cette approximation est mesurée en minimisant la différence entre X et $Z \cdot U$, selon la norme de Frobenius :

$$\|X - ZU\|_F^2 \tag{1.7}$$

La norme de Frobenius est une manière de mesurer l'erreur entre la matrice originale X et la matrice approchée $Z \cdot U$. Elle est définie comme la somme des carrés des différences entre les éléments correspondants des deux matrices (Gillis, 2014) :

$$\|X - ZU\|_F^2 = \sum_{i=1}^{m} \sum_{j=1}^{n} (X_{i,j} - (Z \cdot U)_{i,j})^2 \tag{1.8}$$

Ici chaque utilisateur i est représenté par un vecteur de dimension r (la i-ème ligne de Z), indiquant ses préférences pour les r facteurs latents et chaque film j est représenté par un vecteur de dimension r (la j-ème colonne de U), indiquant sa composition selon les r facteurs latents. Avec $r < \min(m, n)$.
Ensuite l'algorithme s'initialise par des matrices Z et U, souvent avec des valeurs aléatoires non-négatives. Et ensuite il utilise des méthodes itératives pour ajuster les valeurs de Z et U afin de minimiser $\|X - ZU\|_F^2$. Cela peut être fait en alternant les mises à jour de Z et U tout en maintenant la contrainte de non-négativité. En appliquant NMF à une matrice de notes de films, nous pouvons découvrir des patterns latents dans les préférences des utilisateurs et les caractéristiques des films, qui nous permettent donc de faire des recommandations personnalisées et pertinentes.
Voici un exemple qui illustre la factorisation de matrice appliquée à une matrice de notation utilisateur-film de l'exemple précédemment illustré, avec un rang = 2 pour les deux matrices Z et U :

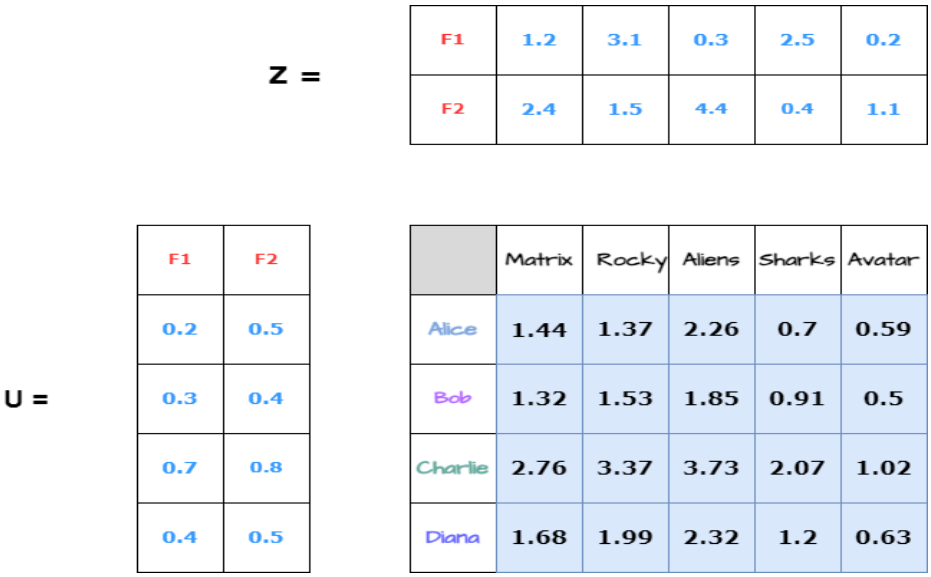


Figure 1.3 NMF - factorisation matricielle

1.4.2 Singular Value Decomposition

Une autre méthode puissante de factorisation de matrice largement utilisée dans les systèmes de recommandations est La Singular Value Decomposition (SVD) (Asoke Nath, 2018) (Bhardwaj, 2023) (Rahul *et al.*, 2021), notamment pour les recommandations de films. La SVD décompose une matrice X en trois matrices distinctes : $X = U\Sigma V^T$ où X est la matrice des notes. U est une matrice orthogonale représentant les utilisateurs dans un espace latent. Σ est une matrice diagonale contenant les valeurs singulières de X qui indiquent l'importance des dimensions latentes, et V^T est la transposée d'une matrice orthogonale représentant les films dans un espace latent. Avant la factorisation, il est courant de centrer les données en soustrayant la moyenne des notes de chaque utilisateur. La matrice X est alors factorisée, et seuls les k plus grandes valeurs singulières et leurs vecteurs associés sont conservés pour réduire le bruit et conserver les dimensions les plus significatives :

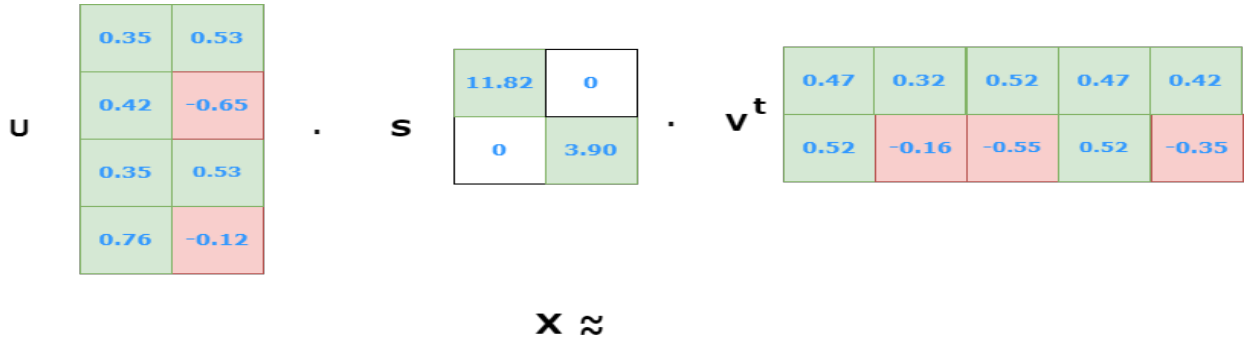
$$X \approx U_k \Sigma_k V_k^T \tag{1.9}$$

Pour prédire la note qu'un utilisateur attribuerait à un film, on calcule le produit scalaire de leurs vecteurs latents :

$$\hat{X}_{i,j} = (U_k \Sigma_k)_i \cdot (V_k^T)_j \tag{1.10}$$

Les films non notés peuvent ensuite être ordonnés selon ces valeurs prédites pour faire des recommandations (Brunton et Kutz, 2019). Les avantages de la SVD incluent la réduction du bruit, l'interprétabilité des

préférences des utilisateurs et des caractéristiques des films, et l'efficacité des calculs après réduction de la dimensionnalité. Voici un exemple qui illustre le fonctionnement de SVD avec l'exemple illustré au début de cette section :



	Matrix	Rocky	Aliens	Sharks	Avatar
Alice	3.02	0.99	1.01	3.02	1.01
Bob	1.02	1.99	3.98	1.02	2.97
Charlie	3.02	0.99	1.01	3.02	1.02
Diana	3.98	2.95	4.93	3.98	3.94

Figure 1.4 SVD - factorisation matricielle

Comme la SVD standard ne gère pas directement les valeurs manquantes et peut être coûteuse en termes de temps et de mémoire pour des matrices très grandes, nous avons fait usage à la librairie Surprise où l'implémentation de SVD gère les valeurs manquantes en utilisant une optimisation par descente de gradient. Elle estime directement les facteurs latents des utilisateurs et des items à partir des notes observées, sans décomposer une matrice complète, ce qui la rend adaptée aux données incomplètes. Voici comment la bibliothèque Surprise calcule une prédiction avec le modèle SVD :

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u \tag{1.11}$$

où :

- $\hat{r}_{ui}$: estimation de la note attribuée par l'utilisateur u à l'item i,
- μ : moyenne globale des notes,
- b_u : biais associé à l'utilisateur u,
- b_i : biais associé à l'item i,

— p_u : vecteur latent représentant l'utilisateur u dans l'espace latent,

— q_i : vecteur latent représentant l'item i dans l'espace latent,

— $q_i^T p_u$: produit scalaire entre p_u et q_i, représentant l'interaction utilisateur-item.

1.4.3 Factorisation de Matrice Booléenne

Un autre type de factorisation matricielle utilisé dans les systèmes de recommandation de films est la factorisation de matrice booléenne (Ignatov *et al.*, 2014). Comme la factorisation matricielle traditionnelle, cette technique permet de décomposer une matrice utilisateur-film en deux matrices de dimensions inférieures, mais la différence ici, c'est au lieu qu'elle utilise des valeurs continues pour capturer les préférences des utilisateurs, la factorisation booléenne travaille avec des valeurs binaires et les opérateurs $\times$ et $+$ sont remplacés respectivement par $\wedge$ et $\vee$. Nous allons maintenant présenter une approche spécifique de la factorisation matricielle booléenne qui est le "Bayesian Boolean Matrix Factorization". L'algorithme "Bayesian Boolean Matrix Factorisation" propose une approche probabiliste pour la factorisation des matrices booléennes à travers la ORMachine. Exactement comme les algorithmes défini précédemment, l'objectif de BMF est également de décomposer une matrice $\mathbf{X}$ en deux matrices binaires $\mathbf{Z} \in \{0,1\}^{N \times L}$ et $\mathbf{U} \in \{0,1\}^{L \times D}$, tels que $\mathbf{X} \approx \mathbf{ZU}$.

Dans cette technique, les multiplications sont remplacés par des opérateurs ET (AND) et les les additions par les opérateurs OU (OR). D'abord, Le modèle génère chaque observation binaire $x_{nd} \in \{0,1\}^D$, ce qui signifie qu'elle ne peut prendre que 1 ou 0 comme valeur, à partir d'un mélange discret de L codes binaires $u_l \in \{0,1\}^D$. Les variables binaires latentes z_{nl} indiquent si le code l est utilisé pour générer l'observation x_{nd}. La probabilité que x_{nd} soit 1 dépasse 1/2 lorsque les codes et les variables latentes sont tous deux égaux à 1 dans au moins une dimension latente ; sinon, elle est inférieure à 1/2. Cette probabilité indique si le code l contribue à cette génération ou pas, et elle est paramétrée par la fonction sigmoïde logistique suivante (Rukat *et al.*, 2017) :

$$\sigma(\lambda) = (1 + e^{-\lambda})^{-1} \text{ où } \lambda \in \mathbb{R}^{+} \tag{1.12}$$

La fonction de vraisemblance est factorisée sur toutes les observations et caractéristiques, utilisant une opération OU entre les codes et les variables latentes pour capturer les dépendances :

$$p(x_{nd}|u, z, \lambda) = \begin{cases} \sigma(\lambda), & \text{si } x_{nd} = \min(1, u_d^T z_n) \\ 1 - \sigma(\lambda), & \text{si } x_{nd} \neq \min(1, u_d^T z_n) \end{cases} \tag{1.13}$$

$$p(x_{nd}|u, z, \lambda) = \sigma\left(\lambda \tilde{x}_{nd}\left(1 - 2\prod_l (1 - z_{nl}u_{ld})\right)\right) \tag{1.14}$$

où $\tilde{x}_{nd} = 2x_{nd} - 1$. Cette expression encode l'opération OU, évaluée à 1 si $z_{nl} = u_{ld} = 1$ pour au moins un l, et à -1 sinon. Le paramètre λ contrôle le bruit : à mesure que $\lambda \to \infty$, toutes les probabilités tendent vers 0 ou 1, transformant le modèle en un produit matriciel booléen déterministe.

Les probabilités à priori Bernoulli indépendants pour u_{ld} et z_{nl} favorisent soit une densité soit une parcimonie dans leurs activations pour permettre au modèle d'apprendre efficacement à partir des données tout en contrôlant la complexité et en promouvant des solutions parcimonieuses ou denses selon le contexte de l'application. La désignation des matrices $\mathbf{U}$ et $\mathbf{Z}$ comme codes et variables latentes est interchangeable, démontrant la symétrie du modèle.

Pour inférer la distribution a posteriori de u_{ld} et z_{nl}, un échantillonneur de Gibbs métropolisé est utilisé, assurant des estimations Monte Carlo à variance réduite. Après chaque itération sur toutes les variables, le paramètre de dispersion λ est mis à jour pour maximiser la vraisemblance, similaire à l'étape M d'un algorithme EM Monte Carlo, en utilisant les prédictions du modèle pour ajuster λ (Rukat *et al.*, 2017).

Les données manquantes sont gérées en ajustant la vraisemblance sur les observations observées. Si $X = (X_{\text{obs}}, X_{\text{mis}})$ décompose la matrice complète en parties observée X_{obs} et manquante X_{mis}, après marginalisation, la vraisemblance $p(X|U, Z, \lambda)$ se réduit à $p(X_{\text{obs}}|U, Z, \lambda)$.

Pour inférer les paramètres U et Z, les données sont représentées comme $\tilde{x}_{nd} \in \{-1, 0, 1\}$, où les valeurs manquantes sont encodées en zéro. Celles-ci contribuent à $\sigma(0) = 1/2$ dans la vraisemblance totale, n'affectant pas la distribution a posteriori sur U et Z. La mise à jour du paramètre de dispersion λ prend en compte le nombre total d'observations manquantes, reflétant la précision des prédictions sur les données observées.

Pour estimer les données manquantes, une estimation Monte Carlo de la distribution prédictive est utilisée comme suit :

$$\frac{1}{S}\sum_{s=1}^{S} p(x_{nd}|U^{(s)}, Z^{(s)}, \hat{\lambda}), \tag{1.15}$$

où $(U^{(s)}, Z^{(s)})$ sont des échantillons a posteriori. Une approximation plus rapide utilise la moyenne a posteriori pour U et Z, $p(x_{nd}|\hat{U}, \hat{Z}, \hat{\lambda})$ (Rukat *et al.*, 2017).

Le Bayesian Boolean Matrix Factorization (BMF) effectue une factorisation de matrice, mais la principale différence réside dans le codage des valeurs. Contrairement aux méthodes de factorisation matricielle qui utilisent directement des valeurs continues. BMF traite les données binaires et utilise un codage spécifique pour représenter les notes continues comme suit :

Il transforme la matrice des notes originale :

	Matrix	Rocky	Aliens	Sharks	Avatar
Alice	3		1		1
Bob	1		4	1	
Charlie	3	1		3	1
Diana		3		4	4

Figure 1.5 B-BMF - factorisation matricielle avant le codage

En une matrice codée avant d'effectuer la factorisation de matrice comme décrit précédemment :

	Matrix	Rocky	Aliens	Sharks	Avatar
Alice	1 1 1 -1 -1	0 0 0 0 0	1 -1 -1 -1 -1	0 0 0 0 0	1 -1 -1 -1 -1
Bob	1 -1 -1 -1 -1	0 0 0 0 0	1 1 1 1 -1	1 -1 -1 -1 -1	0 0 0 0 0
Charlie	1 1 1 -1 -1	1 -1 -1 -1 -1	0 0 0 0 0	1 1 1 -1 -1	1 -1 -1 -1 -1
Diana	0 0 0 0 0	1 1 1 -1 -1	0 0 0 0 0	1 1 1 1 -1	1 1 1 1 -1

Figure 1.6 B-BMF - factorisation matricielle après le codage

1.5 Librairie Surprise

Afin d'implémenter nos algorithmes, nous avons fait usage de la bibliothèque Surprise (Hug, 2020b). C'est une bibliothèque Python spécialisée dans la construction et l'évaluation des algorithmes de recommandation, en particulier ceux du filtrage collaboratif. Cette librairie est un outil puissant pour l'utilisation de ce genre d'algorithmes en Python. Elle offre une large gamme d'algorithmes de recommandation, tels que SVD, KNN, NMF, BaseLine, ainsi que des outils pour gérer les jeux de données de recommandation définis par les utilisateurs soit en chargeant des fichiers CSV, soit en utilisant des dataframes pandas (Wes McKinney,

2010). C'est également possible d'utiliser des larges ensembles de données classiques et réelles tels que les ensembles de données MovieLens, qui sont directement intégrés dans la bibliothèque. Cela permet les utilisateurs de facilement tester leurs algorithmes sur des grands ensembles de données réelles comme le format movielens 100k et movielens 1million. La librairie Surprise permet aussi d'effectuer des évaluations croisées et de générer des prédictions. En facilitant la comparaison entre différents algorithmes, Surprise permet donc d'optimiser les systèmes de recommandation en fonction des besoins spécifiques des utilisateurs. Cette bibliothèque a été conçue pour être utile aux chercheurs qui souhaitent explorer rapidement de nouvelles idées de recommandation en soutenant la création d'algorithmes de prédiction personnalisés, et peut également servir de ressource d'apprentissage pour les étudiants et les utilisateurs moins expérimentés grâce à sa documentation détaillée. Surprise est une ressource très utile dans le domaine de la recommandation.

1.6 Large Language Model (LLM)

CHATGPT :

Le modèle Generative Pre-trained Transformer (GPT), est un des exemples de grand modèle de langage (LLM) développé par OpenAI. Il repose sur des architectures de réseaux de neurones pour générer du texte en se basant sur d'énormes ensembles de données textuelles, souvent composés de millions, voire de milliards d'exemples, afin d'apprendre à comprendre et à produire un langage humain cohérent et contextuellement pertinent. ChatGPT a révolutionné le traitement du langage naturel ces dernières années, en offrant des applications telles que la génération de texte, la traduction automatique et les chatbots. Il est capable de comprendre et de produire du langage humain de manière cohérente et contextuellement pertinente, ce qui en fait un outil puissant pour diverses applications de traitement du langage naturel. Mais malgré que ChatGPT est largement reconnu pour ses capacités exceptionnelles dans la génération de texte et la conversation, il n'est pas encore largement utilisé dans les systèmes de recommandations à part quelques articles qui ont abordé ce sujet de recherche tels que (Liu *et al.*, 2023) et (Palma *et al.*, 2024) qui ont évalué les performances de ChatGPT sur des différents taches de recommandations. Les techniques traditionnelles comme la factorisation matricielle, le filtrage collaboratif et le filtrage de contenu restent dominantes pour prédire les préférences des utilisateurs et fournir des recommandations personnalisées.

Dans ce mémoire et dans le cadre de notre recherche, nous avons eu l'idée d'explorer les capacités de ChatGPT dans les prédictions de systèmes de recommandation dans le domaines des recommandation des films et à travers des oracles que nous créons. Nous avons décidé de le comparer à quelques modèles de

prédiction classiques pour évaluer ses performances. En intégrant ChatGPT dans notre étude, nous cherchons à déterminer son potentiel et ses limites et à voir s'il peut apporter des améliorations par rapport aux méthodes traditionnelles.

1.7 Approches d'ensemble

1.7.1 Bagging

Le bagging, aussi appelé "Bootstrap Aggregating", est une technique couramment utilisée en apprentissage automatique pour améliorer la précision et la robustesse des modèles de prédictions. Il fonctionne en générant plusieurs sous-ensembles de l'ensemble de données d'entraînement en utilisant l'échantillonnage bootstrap, où chaque sous-ensemble est créé en échantillonnant avec remplacement. C'est à dire que que chaque sous-ensemble de l'ensemble d'entraînement est utilisé pour entraîner un modèle indépendant. Les prédictions de tous les modèles entraînés sont ensuite combinées, généralement par vote majoritaire pour les problèmes de classification ou par moyenne pour les problèmes de régression. Cette approche permet de réduire la variance des prédictions et d'atténuer le sur-apprentissage (overfitting), car les erreurs individuelles des modèles sont compensées lorsqu'elles sont agrégées. Mathématiquement, si $f_1(x), f_2(x), \ldots, f_B(x)$ sont les prédictions des B modèles formés, alors la prédiction finale $\hat{f}(x)$ pour une régression est donnée par :

$$\hat{f}(x) = \frac{1}{B} \sum_{b=1}^{B} f_b(x) \tag{1.16}$$

1.7.2 Boosting

Contrairement au bagging traditionnel, où chaque modèle est formé indépendamment, le boosting ajuste les poids des exemples d'entraînement en fonction des erreurs des modèles précédents pour se concentrer sur les exemples difficiles. Après chaque itération d'entraînement, les poids des exemples sont mis à jour pour mettre davantage l'accent sur ceux qui ont été mal prédits. Cette pondération permet aux modèles suivants de se concentrer sur les exemples plus difficiles à prédire. Mathématiquement, si w_i représente le poids de l'exemple i et e_i l'erreur associée, alors les poids peuvent être mis à jour en utilisant une formule telle que $w_i' = w_i \cdot \exp(\epsilon \cdot e_i)$, où ϵ est un paramètre de régulation. Les prédictions des modèles sont ensuite combinées en tenant compte de ces poids. Pour une régression, si $f_1(x), f_2(x), \ldots, f_B(x)$ sont

les prédictions des B modèles et $w_1, w_2, \ldots, w_B$ leurs poids respectifs qui servent à refléter l'importance relative de chaque modèle dans la prédiction finale. Plus un modèle a une faible erreur, plus son poids sera élevé. Donc la prédiction finale $\hat{f}(x)$ est donnée par :

$$\hat{f}(x) = \frac{\sum_{b=1}^{B} w_b \cdot f_b(x)}{\sum_{b=1}^{B} w_b} \tag{1.17}$$

Cette approche permet de réduire encore plus les erreurs persistantes en mettant l'accent sur les exemples difficiles et en améliorant ainsi la robustesse du modèle global.

1.8 Évaluation des algorithmes :

Deux métriques de bases sont souvent utilisées pour mesurer la performance des systèmes de recommandation :

- RMSE (Root Mean Square Error).
- MAE (Mean Absolute Error).

Le RMSE quantifie la différence moyenne entre les valeurs prédites par les modèles et les valeurs réelles des interactions utilisateur-article. Plus précisément, il calcule la racine carrée de la moyenne des carrés des écarts entre les prédictions et les observations (Saadati *et al.*, 2019). Cette métrique est particulièrement utile pour évaluer la précision des prédictions numériques, telles que les évaluations ou les scores prévus par les utilisateurs. La formule mathématique du calcul de l'RMSE est la suivante :

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{1.18}$$

En revanche, le MAE (Mean Absolute Error) mesure l'écart moyen absolu entre les valeurs prédites et les valeurs réelles des interactions utilisateur-article. Contrairement au RMSE, le MAE ne donne pas plus de poids aux grandes erreurs, ce qui en fait une métrique plus robuste aux valeurs aberrantes. La formule mathématique du calcul du MAE est la suivante :

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{1.19}$$

Il existe une grande variété de métriques d'évaluation utilisées pour mesurer la performance des systèmes de recommandation, chacune ayant ses spécificités selon les objectifs du projet. Parmi ces métriques, certaines sont particulièrement adaptées pour préserver le classement des éléments recommandés, telles que la Precision@k qui évalue la précision des recommandations en se basant sur les k premiers éléments suggérés, ou le Mean Reciprocal Rank (MRR) qui mesure la position à laquelle un utilisateur trouve son premier élément pertinent dans la liste des recommandations. Ces métriques sont souvent privilégiées lorsqu'il est important de prendre en compte la qualité du classement des éléments recommandés, ce qui peut être crucial dans des systèmes où l'ordre des suggestions a un impact direct sur l'expérience utilisateur. Toutefois, dans notre projet, nous avons choisi d'utiliser le Root Mean Squared Error (RMSE), une métrique qui permet d'évaluer l'écart entre les évaluations prédites et réelles des utilisateurs. Bien que le RMSE ne soit pas spécifiquement conçu pour mesurer la qualité du classement, il reste une mesure standard de la performance des systèmes de recommandation dans des contextes où l'objectif est de minimiser l'erreur de prédiction. Cette approche s'est avérée pertinente pour notre projet, car elle nous a permis d'évaluer la précision des prédictions sur les scores d'évaluation des films, tout en restant simple et directe à implémenter.

1.9 Ensembles de Données de Référence

Movielens :

Plusieurs ensembles de données sont déjà disponibles gratuitement sur Internet, pour aider les chercheurs et les testeurs à évaluer leurs algorithmes. Notamment, movielens (Harper et Konstan, 2015), une collection de données sur les films et les préférences des utilisateurs, est largement utilisée dans le domaine des systèmes de recommandation. Fournie par le GroupLens Research de l'Université du Minnesota, cette base de données contient des millions de notations et des informations démographiques sur les utilisateurs, publiés pour la première fois en 1998, et qui décrivent les préférences exprimées par les gens pour les films. Ces préférences prennent la forme de tuples (utilisateurs, films, notes, horodatage), chacun étant le résultat d'une personne exprimant une préférence (une note de 1 à 5 étoiles) pour un film à un moment donné. Ces préférences ont été saisies sur le site web movielens (Harper et Konstan, 2015) - un système de recommandation qui demande à ses utilisateurs d'attribuer des notes à des films afin de recevoir des recommandations personnalisées. ce qui permet aux chercheurs de tester les modèles de recommandation

dans des conditions réalistes. Ce qui a beaucoup contribué au développement et à l'amélioration des algorithmes de recommandation, en fournissant une référence solide et fiable pour comparer les performances des modèles (Harper et Konstan, 2015).

1.10 Travaux Connexes :

Il existe plusieurs Frameworks de génération de données synthétiques, les Frameworks permettant de telles simulations sont appelés simulateurs et sont largement utilisés pour former et tester des systèmes de recommandations de différents types (Stavinova *et al.*, 2022b). Un nombre croissant de travaux dans la littérature sur les systèmes de recommandation utilise des données synthétiques et/ou des méthodes de simulation afin de comprendre le comportement des systèmes de recommandation (Ekstrand *et al.*, 2021), voir par exemple (del Carmen Rodríguez-Hernández *et al.*, 2017; Jakomin *et al.*, 2018; Sun et Erath, 2015; Popić *et al.*, 2019; Dankar et Ibrahim, 2021; Nikolenko, 2019; Gonçalves *et al.*, 2020). Il existe de nombreuses utilisations différentes pour de telles données synthétiques comme : préserver la confidentialité des données sous-jacentes (Slokom, 2018), étudier le comportement des méthodes expérimentales dans des conditions contrôlées (Tian et Ekstrand, 2020) et évaluer les algorithmes d'apprentissage par renforcement (Rohde *et al.*, 2018).

Les simulateurs offrent notamment une solution de compromis (en termes de complexité de mise en œuvre, de contrôlabilité et de conformité à la réalité) par rapport aux approches traditionnelles en ligne et également les approches traditionnelles hors lignes, pour la formation et le test des systèmes de recommandation. Ils aident aussi à préserver la confidentialité des données du monde réel et à les compléter/remplacer par des analogues synthétiques, et donnent la possibilité de former et de tester les systèmes de recommandation selon différentes hypothèses concernant les interactions entre les utilisateurs et les systèmes de recommandation (Stavinova *et al.*, 2022a).

La génération de données synthétiques et les techniques de simulation ont été populaires et réussies dans beaucoup d'autres domaines de l'apprentissage automatique tels que la vision par ordinateur (Sankaranarayanan *et al.*, 2018) (Tremblay *et al.*, 2018) et la robotique (Tobin *et al.*, 2018) (Prakash *et al.*, 2020), mais n'ont pas été largement explorées pour les systèmes de recommandation, à l'exception de quelques uns, comme celui des données partiellement synthétiques (Slokom *et al.*, 2020) et SynEvaRec (Provalov *et al.*, 2021) qui ont montré des résultats prometteurs dans le domaine de la recommandation en comparant leurs données générées par leurs simulateurs et les données réelles.

Ces outils de génération de données synthétiques ont donc pour but de créer des ensembles de données

qui imitent des jeux de données réels. Pour ce faire, ils s'appuient sur des données réelles afin de produire des données synthétiques ressemblantes tout en respectant les problématiques de vie privée. Leur objectif est donc de protéger les informations sensibles tout en fournissant des données utiles pour faire leurs analyses et tests, ce qui est différent de notre approche, qui se concentre sur la génération et l'utilisation des données complètement synthétiques, ne nécessitant donc pas un ensemble de données initial réel sur lequel s'appuyer, pour évaluer et améliorer les algorithmes de recommandation.

SynEvaRec (Provalov *et al.*, 2021) est une plate-forme conçue pour non seulement la génération de données synthétiques mais aussi pour l'évaluation des systèmes de recommandation, en testant différents algorithmes de recommandation dans un environnement contrôlé. La plate-forme évalue la performance des algorithmes à l'aide de diverses métriques comme le RMSE et le MAE. En simulant différentes conditions de données, telles que la densité de l'ensemble de données ou la diversité des préférences utilisateur, Elle permet ainsi d'analyser comment ces facteurs influencent les performances des systèmes de recommandation.

Dans (Belletti *et al.*, 2019), par exemple, les chercheurs ont évalué leur méthode de génération de données sur movielens (Harper et Konstan, 2015) 20M en comparant les sommes des notes par film et par utilisateur, ainsi que les valeurs singulières, entre la matrice de données générée et la matrice de données réelle. Ils ont également utilisé des références comme les histogrammes des notes de films pour comparer les données réelles et synthétiques.

Ces techniques ont le potentiel de résoudre des problèmes tels que le manque de grands ensembles de données publiquement disponibles pour la recherche en dehors de l'industrie, comme discuté dans l'introduction. Ce problème existe à cause des préoccupations des entreprises concernant la protection de la confidentialité des utilisateurs et la potentielle divulgation d'indicateurs de performance internes stratégiques. Ces indicateurs incluent des éléments tels que le niveau d'engagement des utilisateurs, leur fidélité au service, ou encore l'évolution du catalogue de produits de l'entreprise au fil du temps. Les données synthétiques, lorsqu'elles reflètent fidèlement les données réelles, offrent aux chercheurs la possibilité d'évaluer leurs méthodes sur des ensembles de données de taille et de complexité comparables à celles utilisées dans les applications commerciales. Cependant, il reste à déterminer dans quelle mesure les données simulées peuvent équilibrer efficacement le compromis entre être suffisamment similaires aux données réelles pour servir de substitut valable, tout en étant suffisamment distinctes pour éviter la divulgation d'informations personnelles sensibles. Les simulateurs peuvent potentiellement générer une quantité infinie de données synthétiques à un coût très faible, ouvrant ainsi de nouvelles opportunités pour la conception des systèmes et des algorithmes de recommandation (Lesnikowski *et al.*, 2021).

Autres travaux :

Dans ce mémoire, nous avons choisi les méthodes de recommandation de base cités ci-dessus, car elles sont couramment utilisées dans les recherches générant des données synthétiques. Elles sont bien documentées dans la littérature et offrent une bonne base pour évaluer les performances des algorithmes dans des scénarios contrôlés. Mais il faut mentionner qu'il existe d'autres approches plus récentes et plus avancés, comme l'approche SLIM (Sparse Linear Methods) et FISM (Factorized Item Similarity Model) qui visent à capturer des relations plus fines entre les utilisateurs et les articles. D'autres modèles avancés incluent les méthodes de filtrage collaboratif neuronal (Neural Collaborative Filtering) et AutoRec, qui utilisent des réseaux de neurones pour apprendre des représentations denses des utilisateurs et des articles, offrant ainsi des recommandations plus personnalisées. Cependant, bien que ces approches sont plus complexes et offrent de grandes promesses en termes de personnalisation et de performance, elles nécessitent souvent des ressources computationnelles plus importantes et des ensembles de données plus vastes pour vraiment démontrer leur efficacité.

CHAPITRE 2
APPROCHE PROPOSÉE

Pour contourner les problèmes abordés dans la section problématique, nous proposons une approche qui se concentre sur l'évaluation et la comparaison de divers algorithmes de recommandation en utilisant différents types d'oracles. L'objectif principal est de déterminer l'efficacité de ces algorithmes dans la prédiction des préférences des utilisateurs et l'amélioration de la précision des recommandations. Nous employons une approche structurée utilisant la bibliothèque Surprise, qui fournit un cadre robuste pour la mise en œuvre et l'évaluation des algorithmes de filtrage collaboratif. Parmi les algorithmes étudiés, on trouve le SVD (Singular Value Decomposition), une technique de factorisation de matrice, le NMF (Non-negative Matrix Factorization), une variante qui impose des contraintes non négatives, le KNN (K-Nearest Neighbors), un algorithme basé sur les similarités entre utilisateurs ou items, le BMF (Bayesian Matrix Factorization), une approche probabiliste de la factorisation de matrice, ainsi que le GPT (Generative Pre-trained Transformer), un modèle avancé utilisant des capacités de traitement du langage naturel. Nous incluons également des techniques de bagging pour évaluer les performances de ces algorithmes en créant des modèles ensemblistes. Les oracles, tels que l'Oracle O-NMF, l'Oracle O-ADA et les jeux de données de la librairie Surprise Dataset, simulent les interactions utilisateur-item, permettant une évaluation sous des conditions contrôlées. L'évaluation principale repose sur l'erreur quadratique moyenne (RMSE), mesurant l'écart entre les prédictions et les notes réelles. Cette section de méthodologie décrira en détail le processus d'évaluation, y compris la préparation des données, la mise en œuvre des algorithmes, l'évaluation des performances et l'analyse des résultats, assurant ainsi la reproductibilité et la transparence de notre démarche.

2.1 Génération des Oracles

Les oracles sont uns des éléments les plus importants dans le cadre de notre projet. Nous avons eu recours à ses derniers afin de générer des données de simulation permettant d'évaluer les performances des algorithmes de recommandation sur différentes conditions et sur des ensembles de données contrôlés. Plus précisément, nous utilisons deux types d'oracles : les Oracles des Arbres de décision Aléatoires (O-ADA) et la factorisation de matrice non négative (O-NMF). Les oracles basés sur les Arbres de décision Aléatoires génèrent des ensembles de données en introduisant des paramètres contrôlables, ce qui permet de simuler différents scénarios de distribution de données. D'autre part, l'oracle NMF se concentre sur la décomposition de matrices pour produire des évaluations approximatives des préférences utilisateur, facilitant ainsi

l'évaluation de la capacité des algorithmes à prédire les notes manquantes. Ces oracles nous permettent de créer des jeux de données variés et réalistes pour évaluer les performances des systèmes de recommandation que nous avons implémentés.

2.1.1 Oracles des Arbres de décision Aléatoires (O-ADA)

Les arbres de décision sont des structures décisionnelles utilisées dans divers contextes pour prédire des résultats basés sur des ensembles de caractéristiques. Dans le cadre des systèmes de recommandation et de notre projet, ils permettent d'estimer les évaluations des utilisateurs pour des films, en fonction de plusieurs attributs (Features). L'idée est de représenter chaque utilisateur par un arbre de décision et associer à chaque film un ensemble de caractéristiques. Ensuite, en fonction de cet arbre, nous pouvons calculer la note finale de chaque utilisateur pour chaque film. Ces arbres sont construits de manière aléatoire et récursive, chaque noeud interne représentant une décision basée sur une caractéristique spécifique du film, tandis que les feuilles de l'arbre représentent les évaluations finales attribuées.

2.1.1.1 Génération des arbres

La génération d'un arbre de décision aléatoire commence par la sélection aléatoire d'une caractéristique parmi l'ensemble k de caractéristiques que nous définissons pour chaque film. Chacun des films est représenté par un ensemble de caractéristiques binaires créées aléatoirement. chaque caractéristique reçoit au hasard une valeur de 1 ou de 0. Ensuite, à chaque niveau de l'arbre, cette caractéristique est utilisée pour séparer les données en branches comme suit :
À partir de la racine, nous déterminons aléatoirement si une feuille doit être attribuée à la branche gauche ou si une sous-branche doit être créée en attribuant une des caractéristique. Ensuite, de manière similaire, nous décidons aléatoirement pour la branche droite si une feuille ou une branche doit y être ajoutée en attribuant une caractéristique. Le processus continue récursivement pour chaque branche, sélectionnant à chaque étape une nouvelle caractéristique pas encore attribuée, jusqu'à ce que les branches atteignent un critère de terminaison, tel qu'une probabilité définie de devenir une feuille ou que toutes les caractéristiques disponibles soient utilisées. Les feuilles sont les points de décision finaux qui attribuent une note à un film basé sur les caractéristiques évaluées le long du chemin de l'arbre.
Le pseudo-code suivant illustre la logique et les étapes décrites précédemment pour la génération d'un arbre de décision aléatoire.

Algorithm 1: OracleRandomTrees

Input: num_users, num_items, num_features, probabilité des feuilles

Output: Matrice complète des notes

Function `create_tree`($features\_list, leaf\_probability$) **:**
    Input : Liste des caractéristiques, probabilité des feuilles;
    Output : Arbre de décision;
    if $features\_list = \emptyset$ *ou* $alatoire(0, 1) < leaf\_probability$ **then**
        Soit N un noeud avec la valeur $N.value = alatoire(NOTE\_MIN, NOTE\_MAX)$;
        return N;
    Créer un noeud N avec la caractéristique aléatoire $N.F \in features\_List$;
    $N.droit \leftarrow$ `create_tree`($features\_list \setminus N.F, leaf\_probability$);
    $N.gauche \leftarrow$ `create_tree`($features\_list \setminus N.F, leaf\_probability$);
    return *N*

Function `calculate_tree_rating`($tree$, $feature\_values$) **:**
    Input : Arbre, valeurs des caractéristiques;
    Output : Note;
    if $tree$ *est une feuille* **then**
        return $tree.value$;
    if *la caractéristique* $tree.F$ *dans* $feature\_values$ *est vrai* **then**
        return `calculate_tree_rating`($tree.droit$, $feature\_values$);
    else
        return `calculate_tree_rating`($tree.gauche$, $feature\_values$);

Initialiser $ratings\_matrix$ comme une matrice de dimensions (num_users, num_items) avec des zéros;
for $user\_id \leftarrow 1$ ***to*** num_users **do**
    user_tree $\leftarrow$ create_tree($features\_list, leaf\_probability$);
    for $item\_id \leftarrow 1$ ***to*** num_items **do**
        $item\_features \leftarrow$ caractéristiques du film item_id // Récupérer les caractéristiques du film;
        $rating \leftarrow$ calculate_tree_rating(user_tree, item_features) // Calculer la note;
        $ratings\_matrix[user\_id][item\_id] \leftarrow rating$ // Remplir la matrice avec la note;

Retourner $ratings\_matrix$

La probabilité de devenir une feuille à un noeud donné est contrôlée par un hyper-paramètre appelé leaf probability qu'on définit avant l'exécution de la fonction. Chaque valeur aléatoire inférieure à la probabilité définie pour devenir une feuille indique que la branche en question se transformera en une feuille et arrêtera la division, fixant ainsi une évaluation finale pour cette branche. Si le noeud ne devient pas une feuille, il continue à se diviser en fonction de la prochaine caractéristique sélectionnée.
Voici un exemple qui illustre la manière dont un arbre aléatoire est généré :

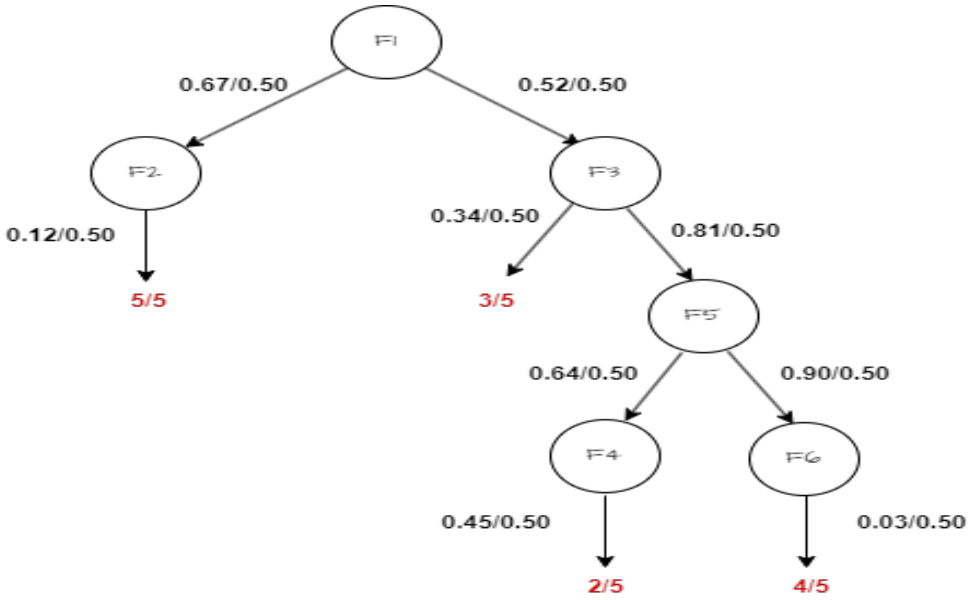


Figure 2.1 Arbres de décision Aléatoires - Génération de l'arbre

Pour illustrer le processus de création d'un arbre aléatoire, considérons l'exemple ci-dessus, avec 6 caractéristiques. Les caractéristiques peuvent être considérées comme un genre, la durée du film, la langue, la popularité, le réalisateur, et les acteurs principaux par exemple. Maintenant supposons que nous fixons une probabilité de 0.5 pour qu'un nœud devienne une feuille, ce qui signifie qu'à chaque étape de la construction de l'arbre, il y a une chance de 50 % que le nœud actuel ne se divise plus et devienne une feuille avec une note aléatoire.

L'arbre commence avec l'ensemble complet des caractéristiques disponibles. Nous commençons à la racine avec la caractéristique (Feature) F1. À gauche de F1, la probabilité de devenir une feuille est de 0.67, ce qui est supérieur à notre seuil de 0.50, donc nous créons un nœud F2. Pour ce nœud F2, la probabilité est de 0.12, inférieure à 0.50, donc F2 devient une feuille avec une note de 5/5. La partie gauche de l'arbre est maintenant terminée. Passons donc à la partie droite de F1, où la probabilité est de 0.52, supérieure à 0.50,

nous créons donc un nouveau nœud F3. Pour le nœud F3, à gauche, la probabilité est de 0.34, inférieure à 0.50, donc nous avons une feuille avec une note de 3/5. À droite de F3, la probabilité est de 0.81, supérieure à 0.50, donc nous créons un nœud F5. À gauche de F5, la probabilité est de 0.64, supérieure à 0.50, donc nous créons un nœud F4. Pour F4, la probabilité est de 0.45, inférieure à 0.50, donc nous avons une feuille avec une note de 2/5. À droite de F5, la probabilité est de 0.90, supérieure à 0.50, donc nous créons un nouveau nœud F5. Pour ce F5, la probabilité est de 0.03, inférieure à 0.50, donc nous avons une feuille avec une note de 4/5.

2.1.1.2 Calcul des notes

Pour calculer la note d'un film basé sur un arbre de décision, nous suivons les branches de l'arbre en fonction des valeurs des caractéristiques (Features) du film comme illustré dans l'exemple suivant :

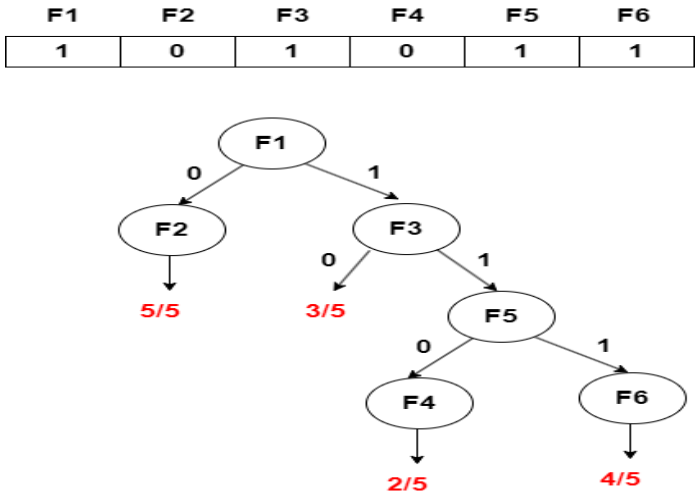


Figure 2.2 Arbres de décision Aléatoires - Calcul de la note finale

Dans cet exemple, les valeurs des caractéristiques du film sont les suivantes :
F1 = 1, F2 = 0, F3 = 1, F4 = 0, F5 = 1, et F6 = 1.
En commençant par la racine de l'arbre, F1, nous prenons la branche correspondant à la valeur 1. Cela nous amène à un nœud pour F3. Ensuite, nous suivons la branche de F3 correspondant à la valeur 1, ce qui nous amène à un nœud pour F5. À partir de F5, nous prenons la branche correspondant à la valeur 1, ce qui nous amène à un nœud pour F6. Ensuite, nous suivons la branche de F6 correspondant à la valeur 1, ce qui nous amène à un nœud pour F4. De F4, nous prenons la branche correspondant à la valeur 0, ce qui nous conduit à une feuille avec une note de 4/5. Ainsi, la note finale pour ce film est 4 sur 5.

2.1.2 Oracle des Factorisations de matrices Non-Négatives

L'Oracle NMF (Non-negative Matrix Factorization), de son côté, utilise une décomposition matricielle pour générer des ensembles de notes aléatoires. Cet oracle repose sur le principe que les interactions entre les utilisateurs et les films peuvent être approximées par le produit de deux matrices de rang inférieur, contenant uniquement des valeurs non négatives.

2.1.2.1 Génération des matrices

$A =$

1.0	1.2	0.8
1.1	1.3	0.9
0.9	1.0	0.7
1.2	1.1	1.0
1..0	1.1	1.2

$B =$

1.2	0.9	1.1	1.3
1.1	1.0	1.2	0.8
0.8	1.1	1.0	1.2

Figure 2.3 O-NMF - Calcul de la note

Lors de l'initialisation, deux matrices, A et B, sont créées comme illustré dans la figure ci-dessus. : A est de dimension (nombre d'utilisateurs, rang) et B de dimension (rang, nombre d'items). Les valeurs des matrices sont initialisées de manière aléatoire, mais bornées entre une valeur minimale et maximale calculées à partir des notes minimales et maximales. Par exemple, supposons un oracle O-NMF avec 5 utilisateurs, 4 items, un rang de 3, des notes minimales de 1.0 et des notes maximales de 5.0. Les matrices A et B sont donc initialisées avec des valeurs aléatoires bornées entre $\sqrt{1.0/3}$ et $\sqrt{5.0/3}$ pour s'assurer que les notes générées par l'oracle seront comprises entre la note minimale et maximale définies au préalable.

2.1.2.2 Calcul des notes

Pour calculer la note d'un utilisateur pour un film, la fonction effectue le produit matriciel de la ligne correspondante de A et de la colonne correspondante de B. Ce produit scalaire donne la prédiction de la note pour l'utilisateur et l'item donnés. Dans la fonction `OracleNMF`, les matrices A et B sont stockées séparément, et la note est calculée à la demande via un produit matriciel. La matrice de notes M est pré-calculée en effectuant le produit matriciel de A et B, permettant ainsi d'accélérer l'accès aux notes pour des matrices de petite taille. Reprenons l'exemple, de l'oracle avec 5 utilisateurs, 4 items, un rang de 3, des notes minimales de 1.0 et des notes maximales de 5.0.

Pour un utilisateur u et un film i, la note est calculée comme suit : Note$_{u,i} = A[u,:] \cdot B[:,i]$.

Ainsi, La figure suivante illustre le calcul d'une note à partir des deux matrices A et B :

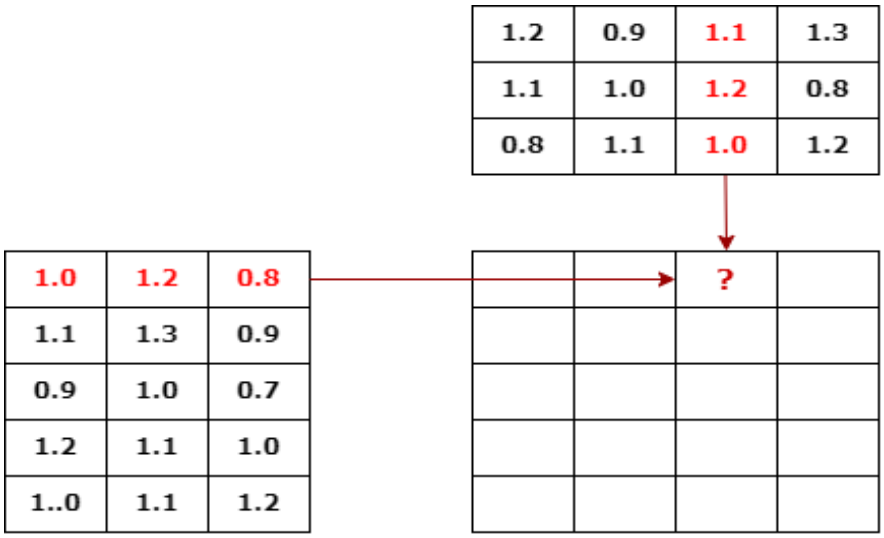


Figure 2.4 O-NMF - Calcul de la note

La note pour l'utilisateur 1 et le film 3 est calculée comme suit :

$$Note_{1,3} = (1.0 \cdot 1.1) + (1.2 \cdot 1.2) + (0.8 \cdot 1.0) = 1.1 + 1.44 + 0.8 = 3.34 = 3/5$$

2.2 Méthodologie des algorithmes utilisés

Avant de détailler les algorithmes de recommandation spécifiques employés dans notre projet, il est d'abord crucial de comprendre la structure de base sur laquelle ces algorithmes sont construits. Pour ce faire, nous avons défini une classe abstraite appelée VirtualAlgo.

2.2.1 Structure de base :

La classe VirtualAlgo sert à construire une base structurée pour toutes les implémentations d'algorithmes et à établir les méthodes fondamentales que chaque algorithme doit implémenter. La classe VirtualAlgo comporte deux méthodes principales : fit et estimate :

fit : La méthode fit est conçue pour être utilisée pour l'entraînement des algorithmes sur un ensemble de données d'entraînement. Elle prend en entrée une liste de tuples (user_id, item_id, rating). Son implémentation spécifique doit être fournie dans les sous-classes dérivées.

estimate : De même, la méthode estimate est censée fournir une estimation de la note pour une paire donnée (u, i) où u est l'identifiant de l'utilisateur et i est l'identifiant de l'article. Cette méthode, tout comme fit, n'est pas implémentée dans VirtualAlgo, mais doit être concrètement définie dans les classes dérivées.

La classe VirtualAlgo fournit donc une architecture qui offre une flexibilité et une structure pour les différents algorithmes de recommandation qui hériteront de cette classe de base.

2.2.2 Implémentation des algorithmes

2.2.2.1 Baseline, KNN, SVD, NMF

Les algorithmes Baseline, KNN, SVD et NMF partagent une structure commune dans leurs méthodes 'fit' et 'estimate', illustrant une approche similaire pour l'entraînement et l'évaluation des modèles de recommandation. Pour chaque algorithme, la méthode 'fit' suit un processus similaire : elle convertit les données d'entraînement en un DataFrame pandas, puis utilise la bibliothèque Surprise pour charger ces données et créer un ensemble d'entraînement complet. Les identifiants bruts des utilisateurs et des articles sont ensuite mappés en identifiants internes, facilitant l'interaction avec les modèles. Après l'entraînement du modèle spécifique à l'algorithme, ce dernier est prêt à faire des estimations.

La méthode 'estimate' vérifie d'abord si les identifiants d'utilisateur et d'article fournis existent dans les identifiants internes. Si un identifiant est absent, il est remplacé par une valeur par défaut (-1) pour les algorithmes Baseline, KNN, et SVD. Pour NMF, si un identifiant est absent, la méthode retourne la moyenne des notes des utilisateurs ou des articles. Ensuite, chaque algorithme utilise sa propre méthode d'estimation pour prédire la note, ajustant si nécessaire les résultats pour garantir qu'ils restent dans les bornes de l'échelle de notation définie.

2.2.2.2 B-BMF

L'algorithme de factorisation des matrices bayésienne (BMF) présente une approche légèrement différente par rapport aux autres algorithmes discutés. Contrairement à Baseline, KNN, SVD et NMF, qui utilisent des processus de conversion et d'entraînement bien définis à l'aide de la bibliothèque Surprise, BMF adopte une méthode plus complexe pour gérer les données et faire des estimations.

La méthode fit commence par la détermination des dimensions du problème en comptant les utilisateurs et les articles à partir des données d'entraînement. Ensuite, elle construit une matrice binaire codée où les notes sont représentées comme des valeurs binaires (-1 ou 1), plutôt qu'une simple échelle de notation. Cette matrice est intégrée dans un modèle bayésien à l'aide de la bibliothèque lom, et une inférence bayésienne est effectuée pour reconstruire la matrice des utilisateurs et des films. Tandis que, les autres algorithmes se contentent de transformer les données d'entraînement en un format compatible et d'utiliser des méthodes d'optimisation pour entraîner les modèles.
Lors de l'estimation, BMF utilise directement la matrice reconstruite pour prédire les notes, calculant la somme des valeurs pertinentes.

2.2.2.3 GPT

Nous avons également eu l'idée d'implémenter une nouvelle méthode utilisant GPT (Generative Pre-trained Transformer) qui adopte une approche radicalement différente par rapport aux autres algorithmes de recommandation. Contrairement à Baseline, KNN, SVD, NMF et B-BMF, qui sont des méthodes basées sur des calculs et des matrices traditionnels, GPT utilise l'API OpenAI pour générer des recommandations.

Tout d'abord, dans la méthode fit, on commence par créer un prompt textuel pour guider le modèle GPT à comment prédire les notes en se basant sur notre message.
Le prompt envoyé à ChatGPT pour le guider à comment prédire est le suivant :
"Étant donné cette matrice d'évaluation utilisateur-élément, prédisez l'évaluation de l'utilisateur et du film donnés et assurez-vous de renvoyer une évaluation qui est uniquement 1.0 , 2.0 , 3.0 , 4.0 ou 5.0, renvoyez uniquement la note dans votre message. votre réponse doit être simplement 1.0 , 2.0 , 3.0 , 4.0 ou 5.0."

Ensuite on transforme notre matrice de notes d'entraînement en format textuel sous forme de phrases décrivant les évaluations des utilisateurs sur les articles avant de les passer à ChatGPT après le premier prompt décrit ci-dessus. On transforme chaque note dans notre matrice de notes en des lignes qui ressemble à : "L'utilisateur {u} a donné au film {i} une note de {note}" Ce prompt est utilisé pour traduire la matrice des notes en texte pour que le modèle GPT puisse les comprendre.

Ensuite, la méthode estimate envoie une requête à l'API GPT, en demandant une prédiction de note pour un utilisateur et un article spécifiques.

la requête que la fonction estimate envoie à ChatGPT est la suivante :
"Pour l'utilisateur {u} et le film {i} quel serait ta prédiction de note ?"
Le modèle GPT retourne ensuite une note prédite sous forme de texte, qui est convertie en nombre flottant.

Cette méthode permet à GPT d'exploiter des capacités de traitement du langage naturel avancées pour générer des recommandations, offrant ainsi une approche novatrice et différente des autres algorithmes, qui reposent sur des calculs internes et des structures de données plus conventionnelles.

2.3 Optimisation des Prédictions

En plus des algorithmes que nous avons présentés ci-dessus, nous avons également développé deux nouvelles méthodes basées sur l'approche de bagging, appelées Bagging et BaggingV2. Ces algorithmes exploitent les modèles de recommandation existants pour améliorer les prédictions en combinant plusieurs instances de ces modèles. Nous allons approfondir le fonctionnement de ces deux méthodes dans les sections suivantes.

2.3.1 Bagging

Comme tous les algorithmes de recommandation utilisent la même interface dans notre cadre d'évaluation, le bagging est relativement simple à mettre en œuvre. Le bagging, qui est l'acronyme de "Bootstrap Aggregating", est une technique qui fonctionne en créant plusieurs instances d'un modèle de base, chacune entraînée sur un sous-ensemble aléatoire des données d'entraînement.
Plus précisément, la méthode fit sélectionne aléatoirement 90% des données pour entraîner chaque modèle, garantissant ainsi que chaque modèle a une perspective légèrement différente des données, tandis que les 10% restants sont ignorés pour chaque itération.
Lors de l'évaluation, la méthode estimate agrège les prédictions de tous les modèles en prenant la médiane des résultats, ce qui permet de réduire la variance et d'améliorer la robustesse des prédictions.

2.3.2 BaggingV2

L'algorithme BaggingV2, quant à lui, suit une approche similaire mais avec une amélioration importante en ajustant dynamiquement les poids des données d'entraînement à chaque itération en fonction des erreurs des prédictions en se basant sur la technique du boosting.

Le boosting est plus complexe que le bagging à mettre en œuvre dans ce contexte car il nécessite que les algorithmes acceptent des observations pondérées, ce qui n'est pas le cas de tous les algorithmes de recommandation. Le boosting ajuste les poids des observations en fonction des erreurs des prédictions précédentes pour améliorer la performance du modèle en combinant plusieurs modèles faibles en un modèle fort. Pour surmonter cette difficulté, nous avons proposé d'adapter le processus de sélection aléatoire des observations du bagging et en tenant compte des poids. Ainsi, une observation avec un poids élevé a une probabilité plus grande d'être sélectionnée, tandis qu'une observation avec un poids faible a une probabilité moindre. Cette méthode permet d'intégrer l'idée du boosting dans les algorithmes qui n'acceptent pas directement les poids, en ajustant la manière dont les échantillons sont choisis pour refléter leur importance dans l'apprentissage.

Dans sa méthode fit, BaggingV2 utilise des poids pour sélectionner les échantillons de données pertinents. Ces poids sont ajustés après chaque itération d'entraînement pour se concentrer davantage sur les instances mal classées. Les échantillons avec des erreurs plus importantes reçoivent des poids plus élevés, ce qui permet aux modèles suivants de se focaliser sur les erreurs précédemment faites. Cette stratégie est conçue pour améliorer la performance globale en réduisant les biais.

La méthode estimate de BaggingV2, comme dans Bagging, calcule la médiane des prédictions des différents modèles, mais bénéficie d'un ajustement dynamique des poids pour mieux capturer les nuances des données. Ces approches de bagging utilisés dans ce travail permettent de combiner les puissances des différents modèles de base et d'améliorer ainsi la qualité globale des prédictions.

2.4 Évaluation des algorithmes

Dans cette section, nous détaillons les méthodes et métriques utilisées pour évaluer les performances des algorithmes implémentés dans notre projet. L'évaluation est effectuée à l'aide d'un script Python qui permet de comparer divers algorithmes en utilisant différents types d'oracles et paramètres de configuration. A part les oracles basés sur des modèles génératifs comme le O-NMF qui est basé sur la factorisation matricielle et O-ADA, basé sur les Arbres de décision Aléatoires, nous avons également utilisé les jeux de données de Movielens : ML-100K et ML-1M, qui sont des ensembles de données de référence disponible dans la librairie Surprise que nous avons utilisé dans le projet.

2.4.1 Description du script

Les principaux éléments de l'évaluation, qui correspondent aux arguments du script Python que nous exécutons, sont les suivants :

- **algorithm :** Les algorithmes que nous voulons évaluer.
- **oracle :** L'ensemble de données à utiliser pour comparer les performances des algorithmes.
- **num_users :** Le nombre d'utilisateurs à générer.
- **num_items :** Le nombre de films à générer.
- **training_percentage :** La proportion des données utilisées pour entraîner l'algorithme.
- **noise_level :** Un niveau de perturbation ajouté aux données.
- **seed :** Le seed assure des résultats identiques à chaque exécution des oracles.
- **rank ou leaf_probability :** Deux Hyper-paramètres nécessaires pour la génération des oracles O-NMF et O-ADA, respectivement.
- **num_tests** Le nombre de tests à effectuer pour chaque combinaison de paramètres afin de calculer la moyenne des résultats.

Lorsque le script Python est exécuté, il procède au calcul des valeurs de RMSE pour chaque algorithme de recommandation sélectionné. Ensuite, il génère des courbes afin de visualiser comment le RMSE varie fonction des différents paramètres que nous avons ajustés. Ces paramètres incluent le rang pour l'algorithme NMF et la probabilité de feuille pour les Arbres de décision Aléatoires, ainsi que les pourcentages d'entraînement et le niveau de bruit pour tous les algorithmes. Les visualisations permettent d'évaluer de manière détaillée l'impact de chaque paramètre sur les performances des algorithmes, facilitant ainsi une comparaison exhaustive des différentes configurations testées.

2.4.2 Usage pratique

Pour utiliser le script d'évaluation, il suffit de lancer la commande suivante dans le terminal :

```
python evalRec.py [OPTIONS]
```

Où `[OPTIONS]` représente les arguments à spécifier pour configurer l'évaluation. Voici les options disponibles avec leurs choix possibles :

— `--oracle` : Type d'oracle à utiliser. Les choix sont : [`nmf`, `random_trees`, `ml-100k`, `ml-1m`]

— `--algorithm` : Algorithmes à évaluer. Les choix sont : [`svd`, `nmf`, `knn`, `base`, `bmf`, `gpt`, `mean`]

— `--seed` : Un nombre entier (Valeur par défaut : 0).

— `--rank` : Un nombre entier pour déterminer le rang pour O-NMF.

— `--leaf_probability` : Un nombre entier entre 1 et 100 pour déterminer la probabilité de feuille.

— `--num_users` : Nombre entier d'utilisateurs à générer pour l'oracle.

— `--num_items` : Nombre entier d'items à générer pour l'oracle.

— `--noise_level` : Un nombre entier (Valeur par défaut : 0).

— `--training_percentage` : Un nombre entier (Valeur par défaut : 90).

— `--num_tests` : Un nombre entier (Valeur par défaut : 1).

Chaque argument peut être ajusté pour adapter l'évaluation selon les besoins.

CHAPITRE 3
EXPÉRIMENTATION

Dans ce chapitre, nous montrons les expérimentations réalisées pour évaluer la performance de nos algorithmes de recommandation. Pour ce faire, nous avons testé nos algorithmes sur deux types de données : les oracles générés (O-NMF et O-ADA) et des ensembles de données réels de Movielens. Les oracles nous permettent de contrôler et de varier les conditions expérimentales, tandis que les jeux de données de Movielens ML 100k et ML 1M nous offrent des bases de référence solides et bien établies dans le domaine des systèmes de recommandation. Ces expérimentations nous permettent d'analyser la robustesse et l'efficacité de nos algorithmes dans des contextes variés et de comparer leurs performances sur des données synthétiques et réelles.
Pour chaque test, une figure illustre les résultats obtenus, ainsi qu'un tableau pour présenter les RMSE de chacune des trois versions de nos algorithmes (D :Défaut , B :Bagging , B2 :BaggingV2). Pour des raisons de lisibilité et de clarté, nous avons décidé de regrouper les valeurs en intervalles de pourcentages au lieu de montrer toutes les 99 lignes individuelles. De plus, les marges d'erreur ont été masquées pour une meilleure lisibilité de la courbe. Nous avons également introduit un bruit gaussien aux données pour simuler des erreurs de mesure ou des imprécisions dans les évaluations des utilisateurs. Le bruit gaussien, est un type de bruit statistique qui suit une distribution normale, caractérisée par une moyenne de zéro et un écart-type déterminé. Les niveaux de bruit choisis sont 0, 1 et 2 qui permettent de simuler différentes perturbations dans les données de manière contrôlée. Le niveau 0 représente l'absence de bruit, le niveau 1 introduit une légère perturbation et le niveau 2 génère un bruit plus significatif, permettant d'observer l'impact du bruit élevé sur les performances des modèles de recommandation. Ces niveaux de bruits permettent de tester la robustesse des algorithmes face à différents types de bruit.

3.1 O-ADA

Nous avons effectué les tests pour évaluer la performance de tous nos algorithmes de recommandation en utilisant des arbres de décision aléatoires comme oracles. Les expérimentations ont été réalisées en faisant varier les pourcentages d'entraînement de 1% à 99% avec une probabilité de feuille de 50% ensuite la probabilité de feuille de 1% à 99% avec un pourcentage d'entraînement de 90%, et le niveau de bruit à trois valeurs différentes : 0, 1, et 2 avec une probabilité de feuille de 50% et deux pourcentages d'entraînement : 90% ensuite 10%.

3.1.1 RMSE VS PROBABILITÉS DE FEUILLE

Script d'exécution Python :

```
python evalRec.py --algorithm svd knn nmf base bmf --oracle random_trees
--num_users 500 --num_items 200 --leaf_probability (1..99) --training_percentage 90
--noise_level 0 --seed 42
```

Algorithmes	SVD			KNN			NMF			BASE			B-BMF		
Versions	D	B	B2	D	B	B2	D	B	B2	D	B	B2	D	B	B2
1-5%	0.40	0.42	0.37	0.71	0.71	0.76	0.66	0.64	0.68	1.15	1.15	1.16	1.71	1.72	1.33
6-15%	0.42	0.44	0.37	0.75	0.76	0.78	0.67	0.64	0.67	1.09	1.09	1.10	1.72	1.75	1.33
16-25%	0.41	0.43	0.34	0.72	0.73	0.75	0.62	0.60	0.63	1.01	1.01	1.03	1.70	1.70	1.33
26-35%	0.38	0.40	0.30	0.65	0.66	0.69	0.58	0.55	0.58	0.94	0.94	0.97	1.69	1.70	1.30
36-45%	0.35	0.36	0.26	0.57	0.59	0.61	0.52	0.49	0.51	0.89	0.89	0.93	1.73	1.70	1.29
46-55%	0.28	0.29	0.21	0.50	0.52	0.54	0.48	0.45	0.45	0.77	0.77	0.81	1.70	1.65	1.27
56-65%	0.19	0.19	0.16	0.43	0.44	0.45	0.43	0.41	0.40	0.67	0.67	0.71	1.64	1.57	1.25
66-75%	0.15	0.15	0.13	0.39	0.40	0.39	0.43	0.41	0.35	0.59	0.59	0.64	1.60	1.51	1.22
76-85%	0.15	0.17	0.09	0.34	0.34	0.32	0.38	0.37	0.30	0.46	0.46	0.50	1.49	1.46	1.17
86-99%	0.16	0.17	0.09	0.22	0.22	0.21	0.26	0.26	0.22	0.28	0.28	0.30	1.47	1.41	1.13

Table 3.1 O-ADA - RMSE vs Probabilités de feuille

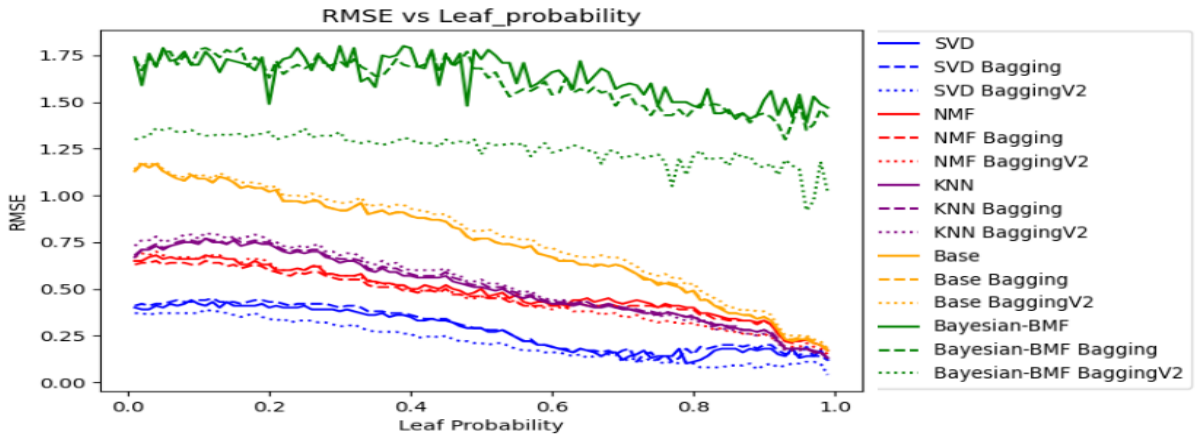


Figure 3.1 O-ADA - RMSE VS Probabilités de feuilles

Observations :

- Comme attendu, tous les algorithmes montrent une amélioration avec l'augmentation de la probabilité des feuilles.
- **SVD** : Meilleure performance globale avec une RMSE variant de 0.44 (B) à 0.09 (B2). La version (BaggingV2) obtient la meilleure RMSE de 0.09, surpassant les versions (Default) et (Bagging).
- **KNN** : Les performances sont légèrement inférieures à celles de SVD vers la fin de l'expérience, avec

une RMSE autour de 0.22. Les deux versions de Bagging n'ont pas apporté d'améliorations.

— **NMF** : Performances globalement meilleures que KNN au début de l'expérience mais moins bonnes que KNN à la fin de l'expérience, avec une RMSE autour de 0.26. La version (BaggingV2) a apporté des améliorations avec une RMSE moyenne de 0.22 ayant les mêmes RMSE que KNN.

— **BASE** : Moins performant que les 3 algorithmes précédents. L'amélioration est notée avec l'augmentation de la probabilité des feuilles. Les versions de Bagging n'ont pas montré des améliorations en termes de RMSE.

— **B-BMF** : Performance globale la plus faible avec des RMSE allant de 1.89 (Bagging pour 1-5%) jusqu'à 1.27 (BaggingV2 pour 66-75% et 86-99%). Cependant, La version (BaggingV2) améliore significativement les performances du modèle avec une RMSE de 1.27, surpassant (Default) et (Bagging) qui ont fini avec des RMSE, respectivement, de 1.59 et 1.60.

3.1.2 RMSE VS POURCENTAGES D'ENTRAÎNEMENTS

Script d'exécution Python :

```
python evalRec.py --algorithm svd knn nmf base bmf --oracle random_trees
--num_users 500 --num_items 200 --leaf_probability 50 --training_percentage (1..99)
--noise_level 0 --seed 42
```

Algorithmes	**SVD**			**KNN**			**NMF**			**BASE**			**B-BMF**		
Versions	D	B	B2	D	B	B2	D	B	B2	D	B	B2	D	B	B2
1-5%	1.14	1.16	1.19	1.29	1.30	1.29	1.03	0.98	1.10	1.23	1.24	1.24	1.89	1.74	1.87
6-15%	0.88	0.89	0.96	0.86	0.86	0.98	0.80	0.76	0.85	0.99	1.00	1.05	1.68	1.60	1.31
16-25%	0.80	0.81	0.81	0.73	0.74	0.84	0.64	0.61	0.75	0.90	0.91	0.96	1.68	1.59	1.29
26-35%	0.65	0.70	0.63	0.66	0.68	0.77	0.56	0.53	0.64	0.86	0.87	0.93	1.64	1.59	1.28
36-45%	0.46	0.52	0.45	0.62	0.63	0.70	0.52	0.50	0.57	0.85	0.85	0.91	1.72	1.61	1.28
46-55%	0.34	0.37	0.33	0.58	0.59	0.65	0.50	0.48	0.53	0.84	0.85	0.90	1.66	1.63	1.28
56-65%	0.30	0.31	0.26	0.55	0.57	0.61	0.49	0.47	0.50	0.84	0.84	0.89	1.72	1.65	1.28
66-75%	0.29	0.29	0.23	0.52	0.54	0.58	0.48	0.47	0.49	0.84	0.84	0.89	1.69	1.62	1.27
76-85%	0.28	0.28	0.21	0.50	0.52	0.55	0.48	0.46	0.48	0.84	0.84	0.89	1.68	1.62	1.28
86-99%	0.26	0.27	0.19	0.47	0.49	0.52	0.47	0.45	0.46	0.84	0.84	0.88	1.59	1.60	1.27

Table 3.2 O-ADA - RMSE vs Pourcentages d'Entraînement

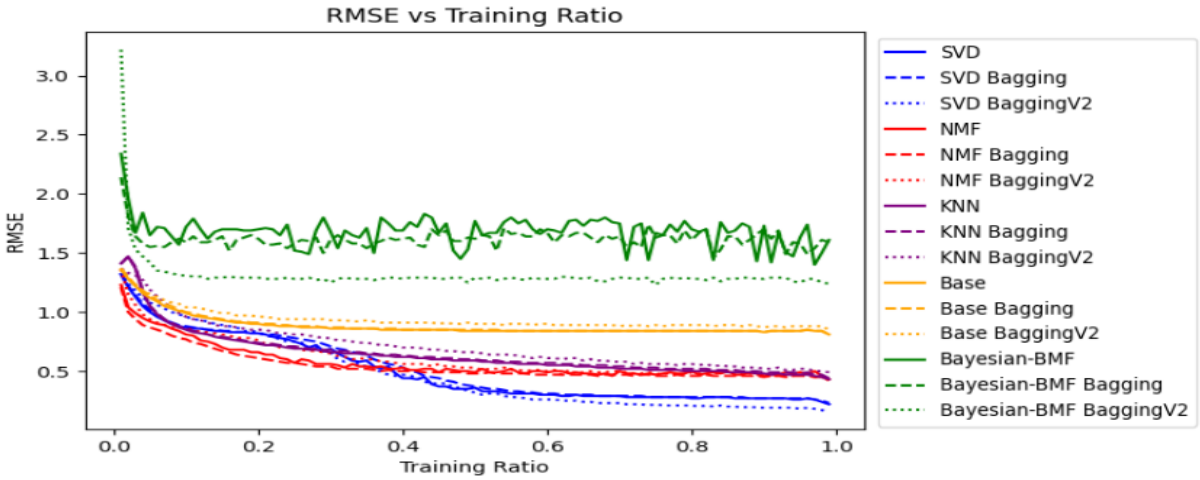


Figure 3.2 O-ADA - RMSE VS Pourcentages d'entraînement

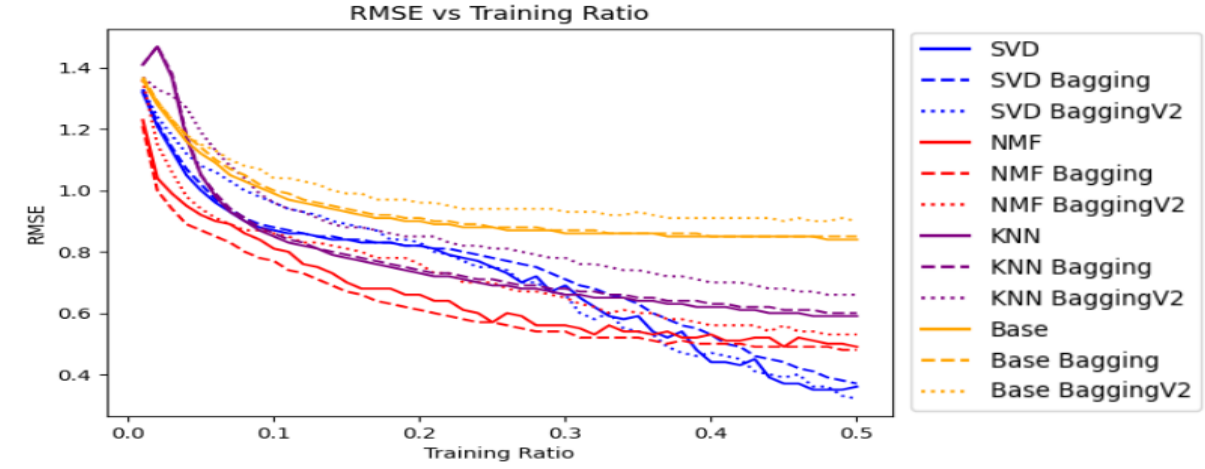


Figure 3.3 O-ADA - RMSE VS Petits Pourcentages d'entraînement

Observations :

- Comme prévu, tous les algorithmes montrent une amélioration avec l'augmentation du pourcentage d'entraînement, jusqu'à un certain niveau.
- **SVD** : Meilleure performance globale, à partir de 40% de données d'entraînement avec une RMSE variant de 1.19 à 0.19. Mais des performances moins bonnes que celles de NMF avec moins de 40% de données d'entraînement. La version (BaggingV2) obtient la meilleure RMSE de 0.19, surpassant les versions (Default) et (Bagging).
- **KNN** : Comportement similaire à NMF, avec une RMSE variant de 1.30 à 0.47. La version (Default) est la meilleure, tandis que les versions (Bagging) et (BaggingV2) montrent des performances très

proches mais sans amélioration.

— **NMF** : Performances globalement meilleures que KNN avec une RMSE variant de 1.10 à 0.45. La version (BaggingV2) obtient une RMSE de 0.45, légèrement meilleure que les versions (Default) et (Bagging) vers la fin de l'expérience.

— **BASE** : Moins performant que les autres algorithmes avec des RMSE variant de 1.24 à 0.84. Les performances s'améliorent, avec l'augmentation du pourcentage d'entraînement, sauf à partir de 26-35% de données d'entraînements les performances sont restées stables avec une RMSE d'autour 0.84 jusqu'à la fin de l'expérience. Les versions de Bagging n'apportent pas de bénéfices par rapport à la version (Default).

— **B-BMF** : Performance globale la plus faible avec des RMSE variant de 1.89 à 1.27. La version (BaggingV2) améliore les performances avec une RMSE de 1.27, surpassant les versions (Default) et (Bagging) qui ont terminé avec des RMSE de 1.59 et 1.60 respectivement.

— De 1 à 5% de données d'entraînement, NMF était le meilleur en termes de RMSE, suivi par SVD, BASE, et enfin KNN. Ensuite de 10 à 30%, KNN a surpassé BASE et SVD, mais NMF est resté encore le plus performant depuis le début. Finalement, Entre 30 et 50%, SVD a commencé à dépasser KNN ensuite NMF, devenant ainsi le meilleur algorithme pour les pourcentages d'entraînement plus élevés.

3.1.3 RMSE VS NIVEAUX DE BRUIT

3.1.3.1 Évaluation avec 90% de données d'entraînement

Script d'exécution Python :

```
python evalRec.py --algorithm svd knn nmf base bmf --oracle random_trees
--num_users 500 --num_items 200 --leaf_probability 50 --training_percentage 90
--noise_level 0 1 2 --seed 42
```

Algorithmes	**SVD**			**KNN**			**NMF**			**BASE**			**B-BMF**		
Versions	D	B	B2	D	B	B2	D	B	B2	D	B	B2	D	B	B2
0	0.27	0.27	0.19	0.48	0.49	0.52	0.46	0.45	0.46	0.83	0.83	0.87	1.74	1.70	1.29
1	0.45	0.44	0.53	0.60	0.61	0.66	0.61	0.61	0.66	0.88	0.88	0.91	1.31	1.20	1.07
2	0.78	0.73	0.86	0.84	0.84	0.94	0.86	0.85	0.92	0.98	0.98	0.98	1.40	1.31	1.16

Table 3.3 O-ADA - RMSE VS Niveaux de Bruit (90% de données d'entraînement)

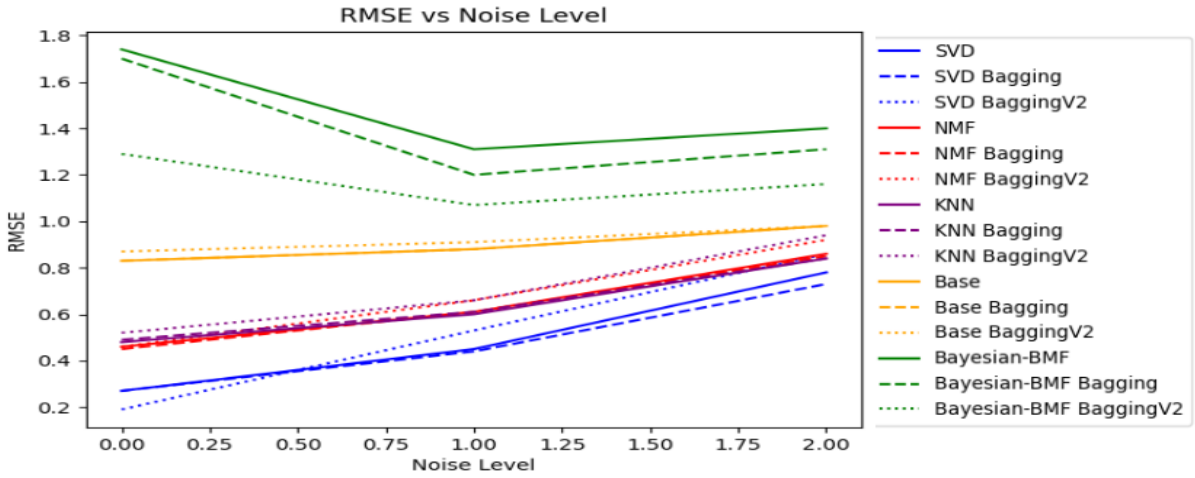


Figure 3.4 O-ADA - RMSE VS Niveaux de bruit (90% de données d'entraînement)

Observations :

- Comme prévu, tous les algorithmes montrent une dégradation de performance avec l'augmentation du niveau de bruit sauf pour B-BMF.
- **SVD** : Meilleure performance globale avec une RMSE variant de 0.19 à 0.73. La version (BaggingV2) obtient la meilleure RMSE, surpassant les versions (Default) et (Bagging) au niveau de bruit 0, ensuite une meilleure performance par la version (Bagging) pour les niveaux de bruit 1 et 2 pour un pourcentage d'entraînement de 90% ensuite 10%.
- **KNN & NMF** : Performances très similaires varient de 0.48 à 0.94. Les performances étaient similaires avec une dégradation de performance avec l'augmentation du bruit. Les versions de Bagging n'ont pas apporté des améliorations significatives tout du long de l'expérience. La version (BaggingV2) a eu les résultats les moins bons par rapport aux deux autres approches.
- **BASE** : Moins performant que les autres algorithmes avec des RMSE variant de 0.83 à 0.98. Les versions de Bagging n'apportent pas d'améliorations significatives, et les performances se détériorent au fur et à mesure que le bruit augmente mais le point fort de BASE réside dans sa bonne gestion de bruit avec des performances stables par rapport aux autres algorithmes.
- **B-BMF** : Performance globale la plus faible avec des RMSE variant de 1.74 à 1.16 avec respectivement +- 0.34 et +- 0.24 de marge d'erreur. Les version (BaggingV2) et (Bagging) améliorent significativement les performances par rapport à la version (Default), mais les performances restent globalement inférieures à celles des autres algorithmes, sauf que B-BMF obtient des meilleures résultats en ajoutant du bruit. Cela prouve qu'il n'est pas stable en termes de performances.

3.1.3.2 Évaluation avec 10% de données d'entraînement

Script d'exécution Python :

```
python evalRec.py --algorithm svd knn nmf base bmf --oracle random_trees
--num_users 500 --num_items 200 --leaf_probability 50 --training_percentage 10
--noise_level 0 1 2 --seed 42
```

Algorithmes	SVD			KNN			NMF			BASE			B-BMF		
Versions	D	B	B2	D	B	B2	D	B	B2	D	B	B2	D	B	B2
0	0.87	0.88	0.96	0.85	0.86	0.96	0.81	0.77	0.86	0.99	1.00	1.04	1.73	1.62	1.31
1	0.94	0.95	1.03	0.97	0.98	1.07	0.98	0.88	0.94	1.06	1.07	1.09	1.51	1.38	1.22
2	1.06	1.07	1.16	1.15	1.15	1.25	1.26	1.05	1.12	1.16	1.17	1.20	1.65	1.51	1.34

Table 3.4 O-ADA - RMSE VS Niveaux de Bruit (10% de données d'entraînement)

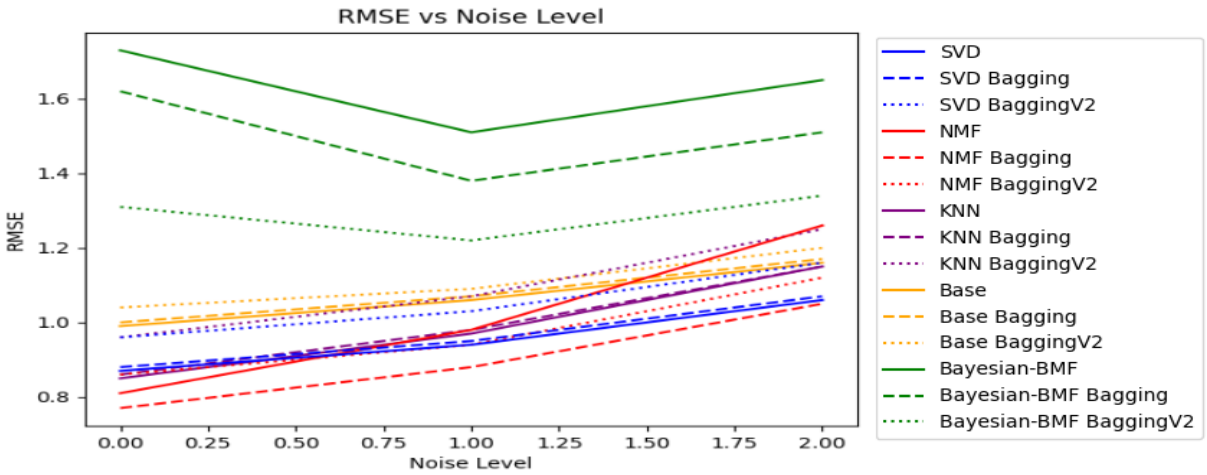


Figure 3.5 O-ADA - RMSE VS Niveaux de bruit (10% de données d'entraînement)

Observations :

- Comme attendu, Des RMSE plus élevés par rapport à ceux obtenus avec un pourcentage d'entraînement de 90%.
- La version (Default) de NMF donne des performances inférieures à celles des algorithmes BASE, KNN et SVD lorsque le bruit est égal à 2 alors qu'il était en deuxième position avec KNN, dans le cas avec 90% de données d'entraînement.
- Par contre, avec 10% de données d'entraînement, la version (Bagging) de NMF obtient les meilleurs résultats pour tous les niveaux de bruit.

3.2 O-NMF

Pour les expérimentations utilisant Oracle-NMF, nous avons également fait varier les variables mais au lieu de la probabilité de feuille, nous avons testé les algorithmes en variant le rang de 1 à 99 avec un pourcentage d'entraînement de 90%. les pourcentages d'entraînement de 1% à 99% avec un rang de 2 et les niveaux de bruit à 0, 1, et 2 avec un rang de 2 et deux pourcentages d'entraînement : 90% ensuite 10%.

3.2.1 RMSE VS RANGS

Script d'exécution Python :

```
python evalRec.py --algorithm svd knn nmf base bmf --oracle nmf --num_users 500
--num_items 200 --rank (1..99) --training_percentage 90 --noise_level 0 --seed 42
```

Algorithmes	SVD			KNN			NMF			BASE			B-BMF		
Versions	D	B	B2	D	B	B2	D	B	B2	D	B	B2	D	B	B2
1-5	0.07	0.07	0.07	0.02	0.02	0.02	0.09	0.08	0.08	0.09	0.09	0.09	0.66	0.62	0.62
6-15	0.04	0.04	0.04	0.03	0.03	0.03	0.08	0.07	0.07	0.04	0.04	0.04	0.52	0.52	0.52
16-25	0.03	0.03	0.03	0.02	0.02	0.02	0.07	0.07	0.06	0.03	0.03	0.03	0.44	0.44	0.44
26-35	0.02	0.02	0.02	0.02	0.02	0.02	0.07	0.06	0.06	0.02	0.02	0.02	0.40	0.40	0.40
36-45	0.02	0.02	0.02	0.02	0.02	0.02	0.07	0.06	0.06	0.02	0.02	0.02	0.40	0.40	0.40
46-55	0.02	0.02	0.02	0.02	0.02	0.02	0.07	0.06	0.06	0.02	0.02	0.02	0.40	0.40	0.40
56-65	0.02	0.02	0.02	0.02	0.02	0.02	0.07	0.06	0.06	0.02	0.02	0.02	0.40	0.40	0.40
66-75	0.02	0.02	0.02	0.01	0.01	0.01	0.07	0.06	0.06	0.02	0.02	0.02	0.40	0.40	0.40
76-85	0.02	0.01	0.01	0.01	0.01	0.01	0.07	0.06	0.06	0.02	0.02	0.02	0.39	0.39	0.39
86-99	0.02	0.01	0.01	0.01	0.01	0.01	0.07	0.06	0.06	0.01	0.01	0.01	0.39	0.39	0.39

Table 3.5 O-NMF - RMSE vs Rangs

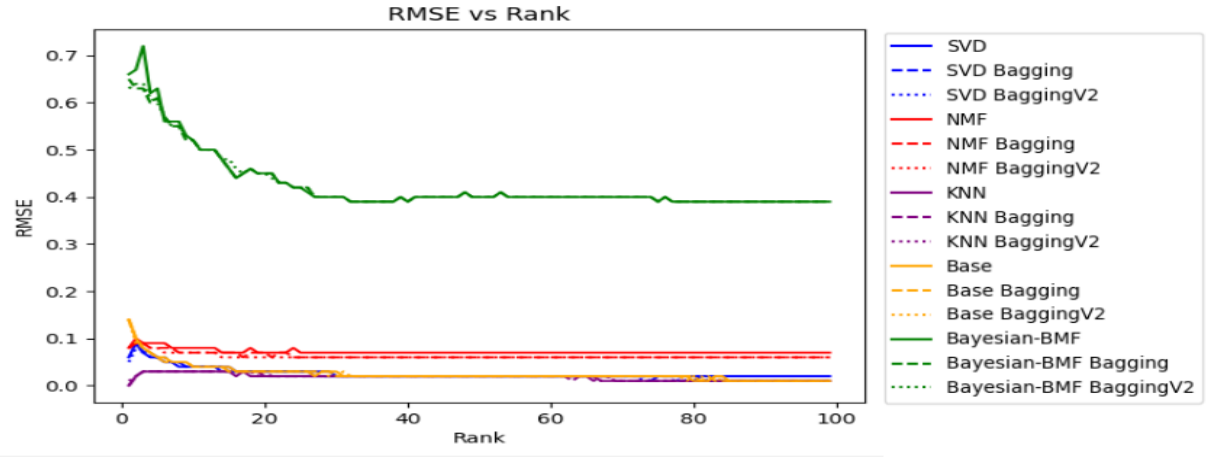


Figure 3.6 O-NMF - RMSE VS Niveaux de rangs

Observations :

- Les algorithmes **SVD**, **KNN** et **Base** ont très bien performé avec toutes les variantes, atteignant des valeurs RMSE aussi basses que 0.01.
- **NMF** a montré des performances un peu inférieures par rapport aux algorithmes SVD, KNN et Base avec une moyenne de RMSE de 0.06.
- Pour l'algorithme **B-BMF**, les résultats ont été globalement moins bons que ceux des autres algorithmes avec un RMSE autour de 0.4.

3.2.2 RMSE VS POURCENTAGES D'ENTRAÎNEMENTS

Script d'exécution Python :

```
python evalRec.py --algorithm svd knn nmf base bmf --oracle nmf --num_users 500
--num_items 200 --rank 2 --training_percentage (1..99) --noise_level 0 --seed 42
```

Algorithmes	SVD			KNN			NMF			BASE			B-BMF		
Versions	D	B	B2	D	B	B2	D	B	B2	D	B	B2	D	B	B2
1-5%	0.33	0.34	0.31	0.37	0.38	0.37	0.18	0.16	0.21	0.39	0.39	0.37	0.86	0.81	1.10
6-15%	0.14	0.14	0.13	0.13	0.14	0.14	0.10	0.09	0.08	0.22	0.23	0.20	0.71	0.67	0.66
16-25%	0.10	0.10	0.10	0.06	0.07	0.07	0.10	0.09	0.09	0.16	0.16	0.14	0.71	0.67	0.66
26-35%	0.10	0.10	0.10	0.04	0.05	0.05	0.10	0.09	0.09	0.13	0.14	0.12	0.69	0.66	0.65
36-45%	0.10	0.10	0.10	0.03	0.04	0.04	0.10	0.10	0.09	0.12	0.12	0.11	0.71	0.65	0.65
46-55%	0.10	0.09	0.09	0.03	0.03	0.03	0.11	0.09	0.09	0.11	0.11	0.10	0.68	0.64	0.65
56-65%	0.09	0.09	0.09	0.03	0.03	0.03	0.10	0.09	0.09	0.11	0.11	0.10	0.67	0.65	0.64
66-75%	0.09	0.09	0.09	0.02	0.02	0.02	0.10	0.10	0.09	0.10	0.11	0.10	0.68	0.64	0.65
76-85%	0.09	0.09	0.09	0.02	0.02	0.02	0.10	0.09	0.09	0.10	0.10	0.10	0.69	0.63	0.64
86-99%	0.09	0.09	0.09	0.02	0.02	0.02	0.10	0.09	0.09	0.10	0.10	0.10	0.67	0.63	0.64

Table 3.6 O-NMF - RMSE vs Pourcentages d'Entraînement

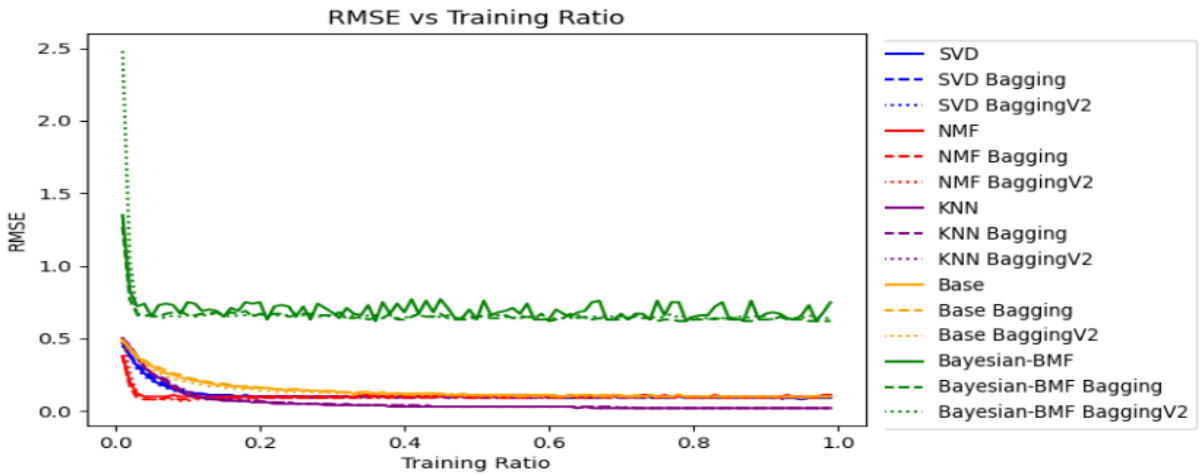


Figure 3.7 O-NMF - RMSE VS Pourcentages d'entraînement

Observations :

- Les variantes de bagging pour les algorithmes **SVD , KNN , NMF et BASE** n'ont pas apporté d'améliorations en termes de RMSE à partir d'un certain niveau. montrant des performances similaires à celles des versions (Default).
- **KNN** avait, globalement, de meilleurs performances à partir de 15% de données d'entraînement surpassant tous les autres algorithmes avec une RMSE de 0.02. NMF avait de meilleures performances pour les pourcentages d'entraînement inférieurs à 15%
- Les Algorithmes **SVD**, **NMF** et **BASE** on fini l'expérience avec des performances similaires et une RMSE moyenne d'autour de 0.09.
- Bayesian-BMF était moins performant que tous les autres algorithmes avec une RMSE moyenne de 0.65 pour toutes ses variantes. Par contre, pour 1%-5% de données d'entraînement la version Bagging était plus efficace que la version par défaut et que BaggingV2
- De 1 à 5% d'entraînement, **NMF** était le meilleur, suivi par **SVD, BASE, KNN et B-BMF**. Entre 6 et 10%, **KNN** a commencé à dépasser progressivement **BASE**, puis **SVD**. Finalement, à partir de 15% et jusqu'à la fin de l'expérience, **KNN** a surpassé tous les autres algorithmes. Les variantes de bagging ont apporté de légères améliorations pour les petits pourcentages d'entraînement.

Ci-dessous, une figure qui clarifie les performances des algorithmes jusqu'à 25% de données d'entraînement :

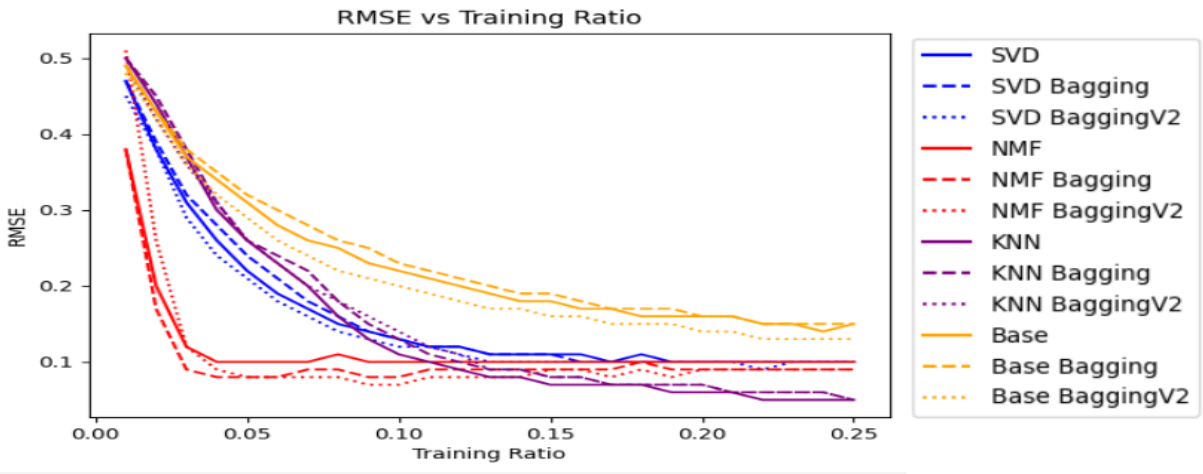


Figure 3.8 O-NMF - RMSE VS Petits Pourcentages d'entraînement

3.2.3 RMSE VS NIVEAUX DE BRUIT

3.2.3.1 Évaluation avec 90% de données d'entraînement

Script d'exécution Python :

```
python evalRec.py --algorithm svd knn nmf base bmf --oracle nmf --num_users 500
--num_items 200 --rank 2 --training_percentage 90 --noise_level 0 1 2 --seed 42
```

Algorithmes	SVD			KNN			NMF			BASE			B-BMF		
Versions	D	B	B2	D	B	B2	D	B	B2	D	B	B2	D	B	B2
0	0.09	0.09	0.09	0.02	0.02	0.02	0.10	0.09	0.09	0.10	0.10	0.10	0.66	0.63	0.64
1	0.19	0.15	0.24	0.23	0.18	0.26	0.17	0.15	0.22	0.15	0.15	0.18	0.70	0.63	0.48
2	0.50	0.33	0.47	0.42	0.36	0.47	0.35	0.30	0.41	0.29	0.29	0.33	0.68	0.67	0.66

Table 3.7 O-NMF - RMSE VS Niveaux de Bruit (90% de données d'entraînement)

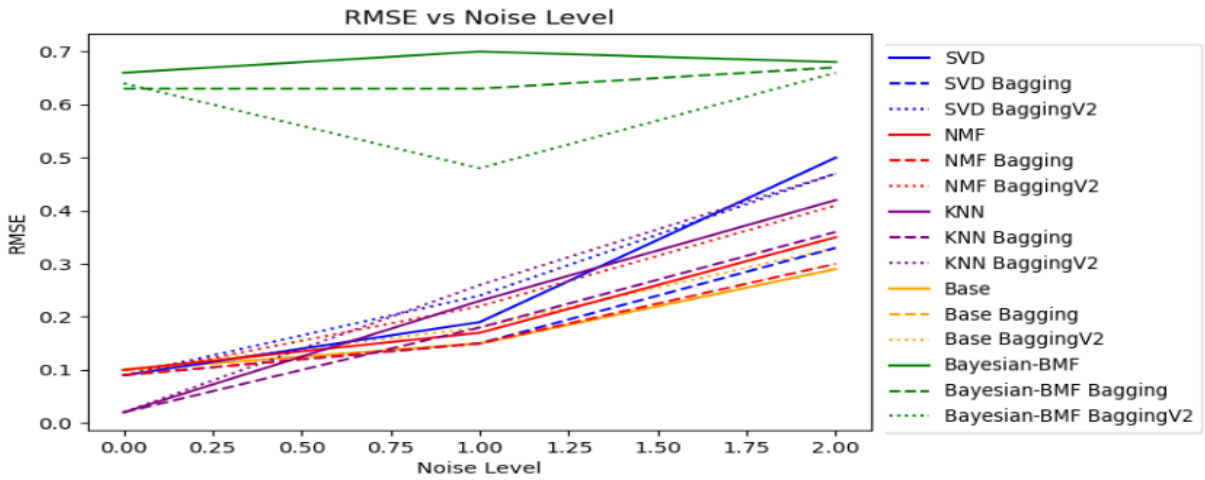


Figure 3.9 O-NMF - RMSE VS Niveaux de bruit (90% de données d'entraînement)

Observations :

- **SVD :** les erreurs RMSE augmentent avec le bruit pour toutes les variantes. La version par défaut passe de 0.09 à 0.50. La version (Bagging) montre des améliorations significative en termes de gestion de bruit par rapport aux approches (BaggingV2) et (Default).
- **KNN :** Commence avec une meilleure RMSE de 0.02 et augmente à 0.42. La première variante de Bagging montre des améliorations significatives au niveau des RMSE par rapports aux deux autres variantes (BaggingV2) et (Default).
- **NMF :** Montre une tendance similaire à KNN avec des augmentations de RMSE de 0.10 à 0.41. Le (Bagging) présente des performances meilleures que les versions (Default) et (Bagging).
- **BASE :** Termine l'expérience avec la meilleure RMSE, atteignant 0.29. La version (BaggingV2) a légèrement diminué les performances, tandis que les versions (Bagging) et (Default) avaient exactement les mêmes valeurs de RMSE tout au long de l'expérience.
- **B-BMF :** Démarre avec une RMSE plus élevée à 0.66 avec +- 0.12 de marge d'erreur, diminue à 0.48 avec +- 0.09 de marge d'erreur pour sa version (BaggingV2) et pour bruit = 1. Ensuite obtient une RMSE égale à 0.67 avec +- 0.13 de marge d'erreur pour toutes les variantes du modèle pour bruit = 2. Les variantes de bagging ont montré d'amélioration significative sur tout la version (BaggingV2) avec le niveau de bruit égale à 1.

3.2.3.2 Évaluation avec 10% de données d'entraînement

Script d'exécution Python :

```
python evalRec.py --algorithm svd knn nmf base bmf --oracle nmf --num_users 500
--num_items 200 --rank 2 --training_percentage 10 --noise_level 0 1 2 --seed 42
```

Algorithmes	SVD			KNN			NMF			BASE			B-BMF		
Versions	D	B	B2	D	B	B2	D	B	B2	D	B	B2	D	B	B2
0	0.13	0.13	0.12	0.11	0.13	0.14	0.10	0.08	0.07	0.22	0.23	0.20	0.74	0.67	0.65
1	0.27	0.26	0.31	0.33	0.32	0.39	0.51	0.30	0.32	0.30	0.30	0.32	0.81	0.78	0.59
2	0.44	0.43	0.50	0.52	0.49	0.62	0.89	0.50	0.54	0.44	0.44	0.48	1.02	0.94	0.76

Table 3.8 O-NMF - RMSE VS Niveaux de Bruit (10% de données d'entraînement)

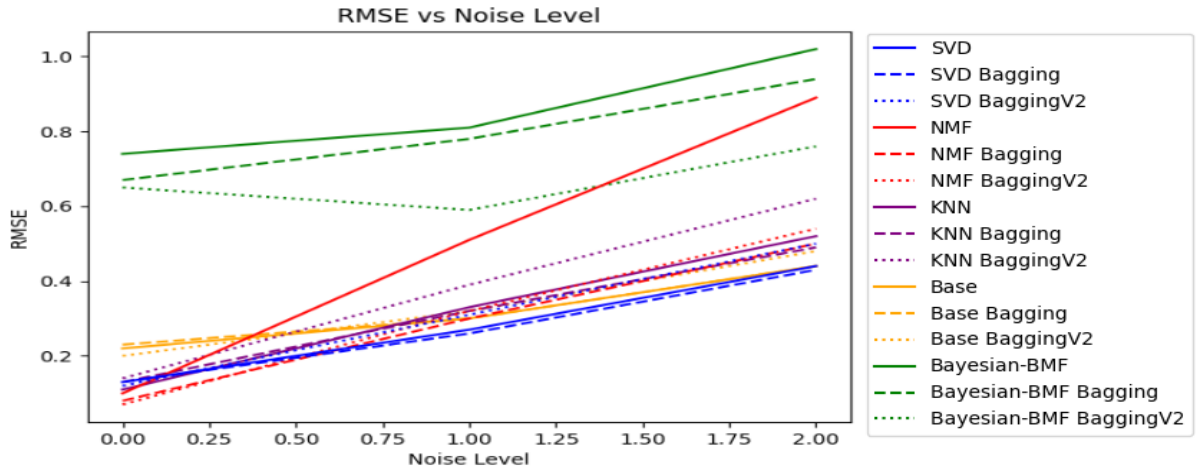


Figure 3.10 O-NMF - RMSE VS Niveaux de bruit (10% de données d'entraînement)

Observations :

- **SVD :** Avec 10% de données d'entraînement, SVD a évité l'avant-dernière position et a obtenu la première place pour bruit = 1 et bruit = 2.
- **BASE :** A mal commencé l'expérience pour un bruit égale à 0 mais il a obtenu des résultats proches de KNN, NMF et SVD pour un bruit égale à 1. Et il a obtenu exactement le mêmes résultat que SVD pour bruit = 2.
- **KNN :** A perdu sa première place pour bruit = 0, mais a conservé le même classement pour le reste de l'expérience.
- **NMF :** Était en deuxième position parmi les versions (Default) du restes des algorithmes, avec 90% d'entraînement, mais pour bruit = 1 et bruit = 2 et 10% de données d'entraînement, ses résultats sont

désormais loin derrière ceux de SVD, BASE et KNN. Les performances sont également inférieures à celles de (BaggingV2) B-BMF.

— **B-BMF :** Les deux variantes de bagging ont amélioré les résultats. Et cette fois-ci, avec 10% de données d'entraînement, la version BaggingV2 a permis de surpasser NMF.

3.3 MOVIELENS 100K

Pour les expérimentations avec l'ensemble de données ML-100k, nous avons fait varier les pourcentages d'entraînement de 1% à 99% et les niveaux de bruit de 0, 1, et 2 pour un pourcentage d'entraînement de 90% ensuite 10%.

3.3.1 RMSE VS POURCENTAGES D'ENTRAÎNEMENTS

Script d'exécution Python :

```
python evalRec.py --algorithm svd knn nmf base bmf --oracle ml-100k
--training_percentage (1..99) --noise_level 0 --seed 42
```

Algorithmes	**SVD**			**KNN**			**NMF**			**BASE**			**B-BMF**		
Versions	D	B	B2	D	B	B2	D	B	B2	D	B	B2	D	B	B2
1-5%	1.06	1.05	1.07	1.12	1.11	1.08	1.24	1.20	1.20	1.06	1.06	1.07	3.10	3.23	3.47
6-15%	1.00	1.00	1.02	1.09	1.09	1.03	1.12	1.06	1.08	1.01	1.01	1.02	1.93	1.84	2.44
16-25%	0.98	0.97	0.99	1.00	1.00	1.01	1.06	1.00	1.03	0.98	0.98	0.99	1.74	1.64	1.49
26-35%	0.96	0.96	0.98	0.97	0.97	0.99	1.03	0.98	1.00	0.97	0.97	0.98	1.69	1.62	1.44
36-45%	0.96	0.95	0.97	0.96	0.96	0.97	1.01	0.96	0.99	0.96	0.96	0.97	1.66	1.59	1.42
46-55%	0.95	0.95	0.96	0.95	0.95	0.97	1.00	0.95	0.98	0.95	0.95	0.96	1.63	1.58	1.39
56-65%	0.95	0.94	0.95	0.95	0.94	0.96	0.98	0.95	0.97	0.95	0.95	0.96	1.61	1.57	1.36
66-75%	0.94	0.94	0.95	0.94	0.94	0.96	0.97	0.94	0.96	0.95	0.95	0.95	1.60	1.55	1.33
76-85%	0.93	0.93	0.94	0.93	0.93	0.95	0.97	0.94	0.96	0.94	0.95	0.95	1.59	1.54	1.31
86-99%	0.92	0.93	0.94	0.93	0.93	0.95	0.96	0.94	0.96	0.94	0.95	0.95	1.58	1.53	1.29

Table 3.9 ML-100K - RMSE vs Pourcentages d'Entraînement

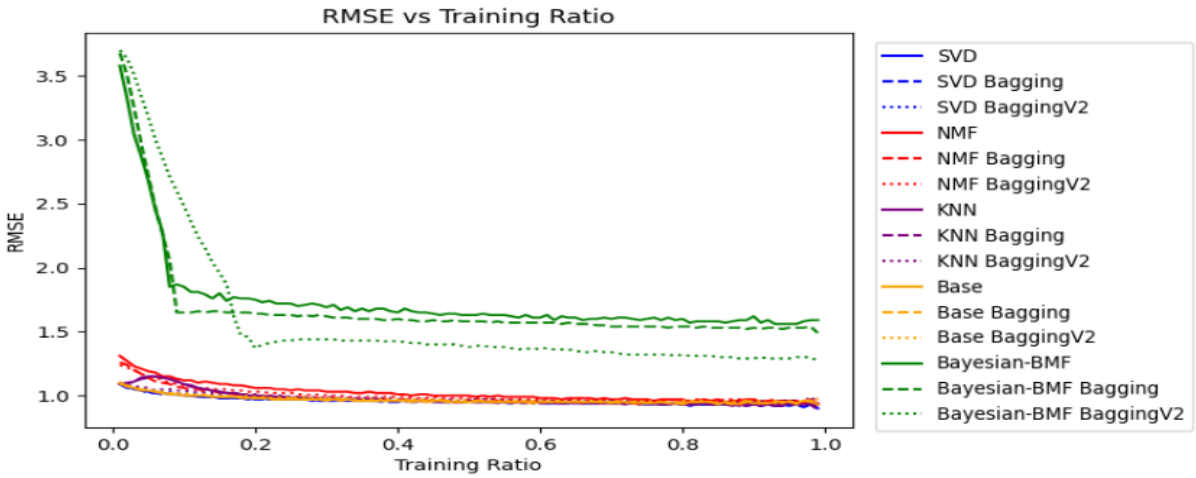


Figure 3.11 ML-100K - RMSE VS Pourcentages d'entraînement

Observations :

- **SVD & KNN & NMF & BASE :** Résultats similaires à partir de 20% avec des légères différences en termes de RMSE. tous les algorithmes on fini l'expérience avec une RMSE moyenne égale à 0.94.
- Les versions bagging de tous les algorithmes n'ont pas amélioré les performances globales sauf la version (BaggingV2) pour **Bayesian-BMF**, qui a montré une amélioration notable au niveau des RMSE à partir de 20% de données d'entraînement et finissant l'expérience avec une valeur RMSE de 1.29 alors que pour les deux autres versions la RMSE était autour de 1.55 Mais malgré cette amélioration, il a quand même eu des résultats moins bon que les autres algorithmes.

3.3.2 RMSE VS NIVEAUX DE BRUIT

3.3.2.1 Évaluation avec 90% de données d'entraînement

Script d'exécution Python :

```
python evalRec.py --algorithm svd knn nmf base bmf --oracle ml-100k
--training_percentage 90 --noise_level 0 1 2 --seed 42
```

Algorithmes	SVD			KNN			NMF			BASE			B-BMF		
Versions	D	B	B2	D	B	B2	D	B	B2	D	B	B2	D	B	B2
0	0.93	0.92	0.93	0.93	0.93	0.95	0.95	0.93	0.96	0.94	0.94	0.95	1.52	1.51	1.19
1	0.96	0.95	0.98	0.97	0.96	0.99	0.99	0.95	1.00	0.96	0.96	0.97	1.19	1.19	1.06
2	1.07	1.01	1.07	1.05	1.03	1.08	1.08	1.00	1.08	1.01	1.01	1.02	1.24	1.24	1.16

Table 3.10 ML-100K - RMSE VS Niveaux de Bruit (90% de données d'entraînement)

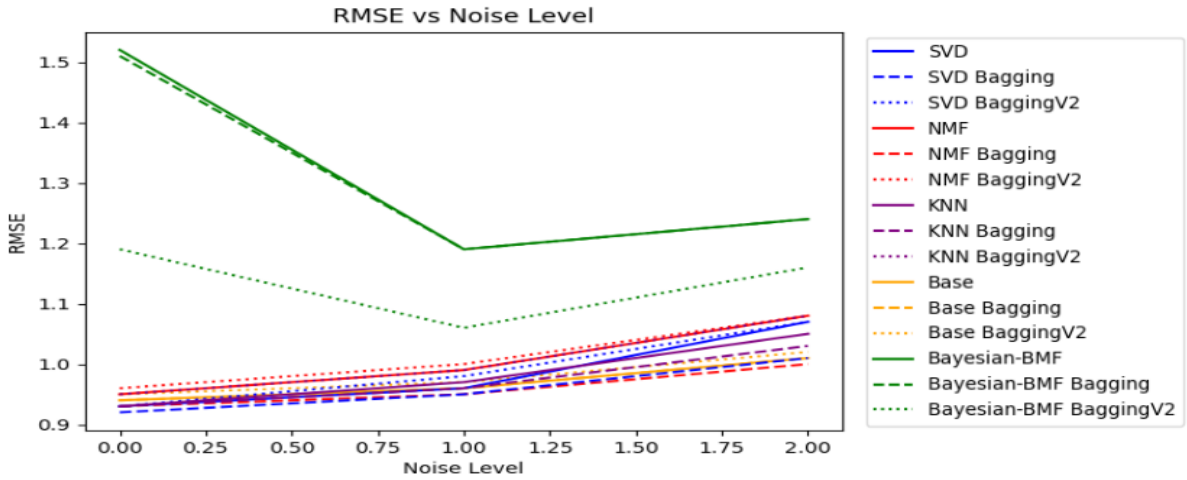


Figure 3.12 ML-100K - RMSE VS Niveaux de bruit (90% de données d'entraînement)

Observations :

- **SVD & KNN & NMF & BASE :** Des résultats plus ou moins similaires avec une RMSE moyenne d'environ 0.94 pour un niveau bruit égale à 1. Ensuite une RMSE moyenne d'environ 0.95 pour le niveau 2 de bruit, et finalement une RMSE moyenne de 1.05 pour un bruit de 2.
- **Bayesian-BMF :** Un comportement similaire aux autres expériences précédentes face au bruit, avec une meilleure performance en ajoutant du bruit et une grande marge d'erreur d'environ +- 0.27.
- La variante (Bagging) a montré des améliorations des performances pour SVD , KNN et NMF. tandis que la variante (BaggingV2) a montré des améliorations des performances que pour B-BMF et aucune amélioration pour BASE.

3.3.3 RMSE VS NIVEAUX DE BRUIT

3.3.3.1 Évaluation avec 10% de données d'entraînement

Script d'exécution Python :

```
python evalRec.py --algorithm svd knn nmf base bmf --oracle ml-100k
--training_percentage 10 --noise_level 0 1 2 --seed 42
```

Algorithmes	SVD			KNN			NMF			BASE			B-BMF		
Versions	D	B	B2	D	B	B2	D	B	B2	D	B	B2	D	B	B2
0	1.00	1.00	1.01	1.10	1.10	1.03	1.12	1.06	1.08	1.00	1.00	1.01	1.68	1.66	1.49
1	1.03	1.02	1.04	1.17	1.17	1.08	1.24	1.14	1.15	1.03	1.03	1.04	1.50	1.44	1.43
2	1.09	1.08	1.09	1.32	1.31	1.16	1.41	1.28	1.27	1.08	1.08	1.08	1.74	1.63	1.58

Table 3.11 ML-100K - RMSE VS Niveaux de Bruit (10% de données d'entraînement)

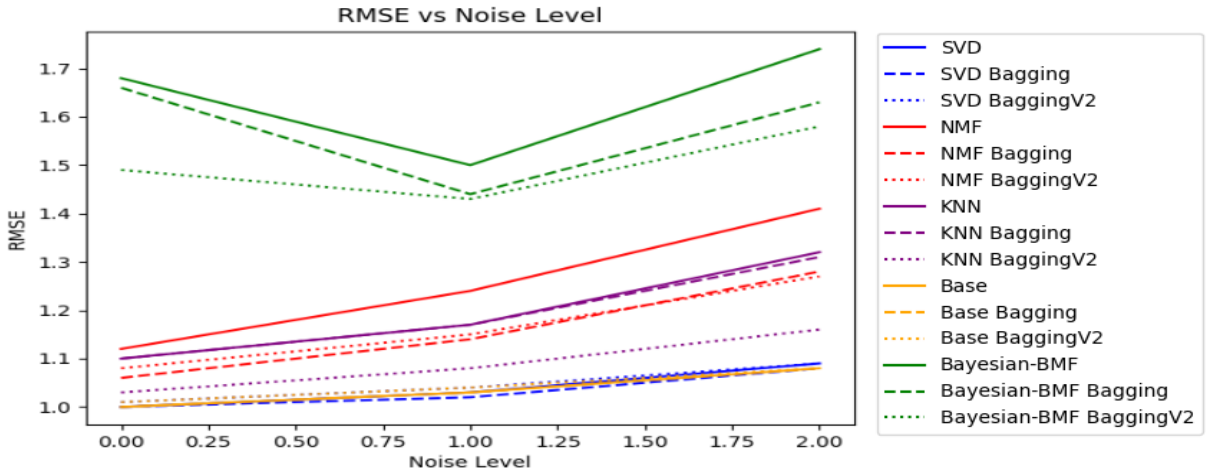


Figure 3.13 ML-100K - RMSE VS Pourcentages d'entraînement (10% de données d'entraînement)

Observations :

- **SVD, KNN, NMF, BASE :** SVD et BASE ont montré des résultats similaires et ont mieux performé que tous les autres algorithmes, avec 10% de données d'entraînement. KNN et NMF suivent, tandis que BMF reste en dernière position.
- **Bayesian-BMF :** Le baggingv2 a encore amélioré les performances, et cette fois-ci, le bagging a également eu un effet positif.
- Pour SVD et BASE, les versions de bagging n'ont eu aucun effet significatif sur les RMSE. En revanche, pour NMF, les deux variantes de bagging ont amélioré les résultats, et pour KNN, seule la version (BaggingV2) a eu un effet positif.

3.4 MOVIELENS 1M

Pour les expérimentations avec l'ensemble de données ML-1M, nous avons également fait varier les pourcentages d'entraînement de 1% à 99% et les niveaux de bruit de 0, 1, et 2 pour un pourcentage d'entraînement de 90% ensuite 10%.

3.4.1 RMSE VS POURCENTAGES D'ENTRAÎNEMENTS

Script d'exécution Python :

```
python evalRec.py --algorithm svd knn nmf base bmf --oracle ml-1m
--training_percentage (1..99) --noise_level 0 --seed 42
```

Algorithmes	SVD			KNN			NMF			BASE			B-BMF		
Versions	D	B	B2	D	B	B2	D	B	B2	D	B	B2	D	B	B2
1-5%	0.99	0.99	1.01	1.08	1.08	1.02	1.14	1.09	1.11	1.00	1.00	1.01	2.19	2.13	2.67
6-15%	0.94	0.94	0.95	0.97	0.97	0.97	1.02	0.96	0.99	0.94	0.95	0.95	1.67	1.59	1.41
16-25%	0.93	0.92	0.93	0.93	0.93	0.94	0.97	0.93	0.95	0.93	0.93	0.93	1.59	1.55	1.34
26-35%	0.92	0.91	0.93	0.93	0.92	0.93	0.95	0.92	0.94	0.92	0.92	0.93	1.56	1.54	1.30
36-45%	0.91	0.91	0.92	0.92	0.91	0.93	0.94	0.92	0.94	0.91	0.92	0.92	1.55	1.53	1.27
46-55%	0.90	0.89	0.90	0.92	0.91	0.93	0.93	0.92	0.93	0.91	0.91	0.92	1.54	1.52	1.25
56-65%	0.89	0.88	0.89	0.91	0.91	0.93	0.93	0.92	0.93	0.91	0.91	0.92	1.53	1.52	1.24
66-75%	0.88	0.87	0.88	0.91	0.90	0.92	0.92	0.92	0.92	0.91	0.91	0.92	1.52	1.51	1.23
76-85%	0.87	0.87	0.88	0.90	0.90	0.92	0.92	0.91	0.92	0.91	0.91	0.91	1.51	1.51	1.22
86-99%	0.86	0.86	0.87	0.90	0.89	0.92	0.92	0.91	0.92	0.91	0.91	0.91	1.51	1.50	1.21

Table 3.12 ML-1M - RMSE vs Pourcentages d'Entraînement

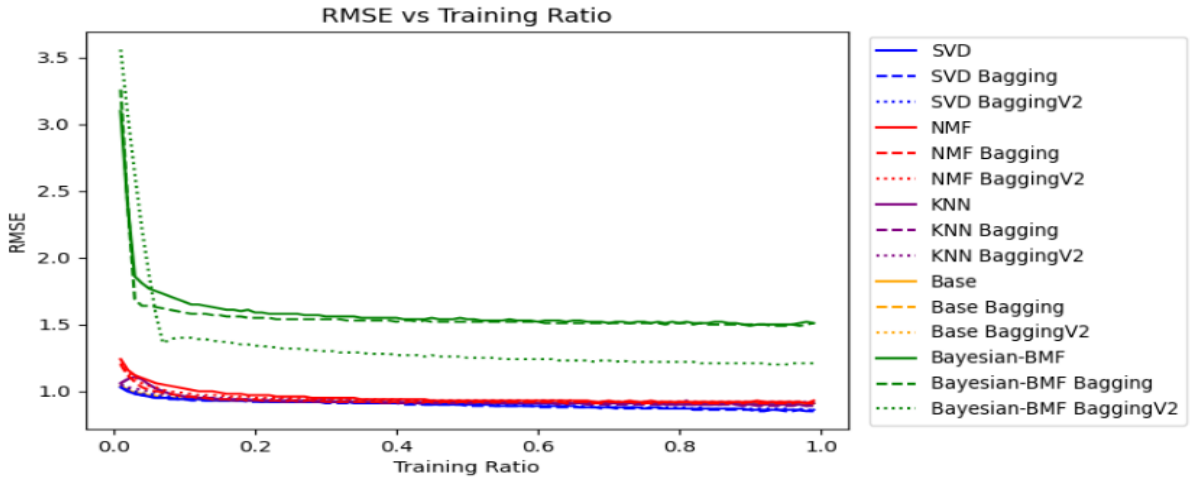


Figure 3.14 ML-1M - RMSE VS Pourcentages d'entraînement

Observations :

- **SVD :** Meilleures performances en termes de RMSE avec une RMSE de 0.86 à la fin de l'expérience.
- **BASE & NMF & KNN :** Performances similaires avec une RMSE d'autour 0.91.
- Les versions bagging de tous les algorithmes n'ont pas amélioré les performances globales sauf la version (BaggingV2) pour Bayesian-BMF, qui a montré une amélioration notable au niveau des RMSE avec une valeur de 1.21 alors que pour les deux autres versions l'RMSE était autour de 1.5. Mais malgré

cette amélioration, il a quand même eu des résultats moins bon que les autres algorithmes.

3.4.2 RMSE VS NIVEAUX DE BRUIT

3.4.2.1 Évaluation avec 90% de données d'entraînement

Script d'exécution Python :

```
python evalRec.py --algorithm svd knn nmf base bmf --oracle ml-1m
--training_percentage 90 --noise_level 0 1 2 --seed 42
```

Algorithmes	SVD			KNN			NMF			BASE			B-BMF		
Versions	D	B	B2	D	B	B2	D	B	B2	D	B	B2	D	B	B2
0	0.86	0.86	0.87	0.90	0.89	0.92	0.92	0.91	0.92	0.90	0.91	0.91	1.48	1.46	1.13
1	0.92	0.89	0.93	0.94	0.93	0.95	0.93	0.92	0.94	0.92	0.92	0.93	1.15	1.13	1.00
2	1.04	0.97	1.04	1.03	1.01	1.04	0.98	0.96	1.01	0.97	0.97	0.99	1.16	1.15	1.11

Table 3.13 ML-1M - RMSE VS Niveaux de Bruit (90% de données d'entraînement)

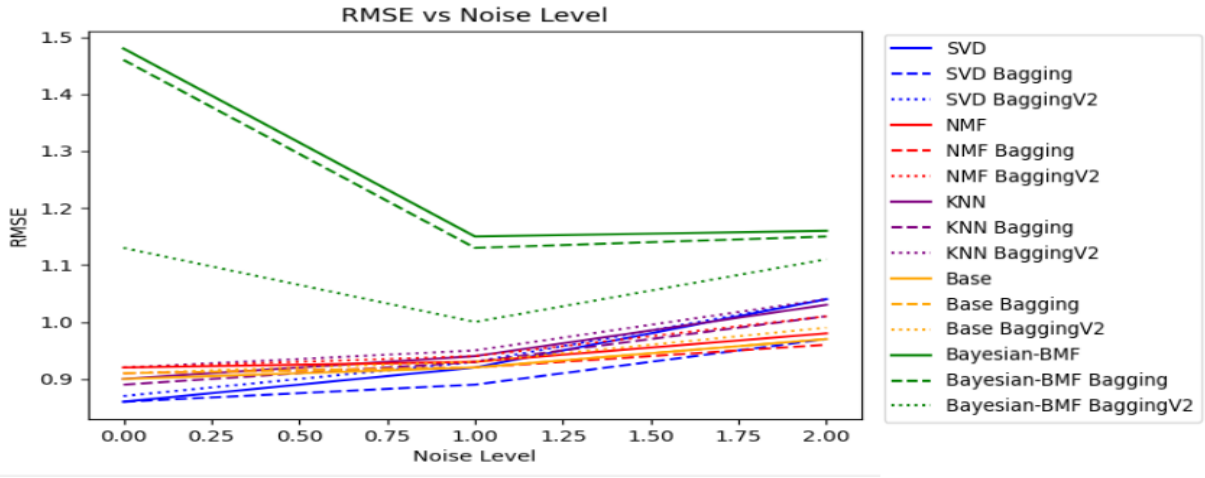


Figure 3.15 ML-1M - RMSE VS Niveaux de bruit (90% de données d'entraînement)

Observations :

— **NMF & BASE :** Des performances similaires avec une bonne stabilité face au bruit. Les performances ont un peu diminué avec l'augmentation du bruit allant de 0.92 à 0.93 à 0.98 pour NMF et de 0.90 à 0.92 à 0.97 pour BASE pour, respectivement, les niveaux de bruit 0 1 et 2 pour un pourcentage d'entraînement de 90% ensuite 10%. Les deux approches de Bagging n'ont pas apporté d'améliorations significatives.

— **KNN :** L'algorithme KNN a également bien géré le bruit avec des RMSE allant de 0.90 à 0.94 à 1.03. La version de (BaggingV2) a apporté une légère amélioration significative par rapport à (Default).

— **SVD :** Globalement, SVD a le mieux performé pour les 3 niveaux de bruits allant de 0.86 à 0.92 à 1.04 pour la version (Default). Sa version (BaggingV2) a amélioré significativement les prédictions par rapport à la version par (Default) allant de 0.86 à 0.89 à 0.97.

— **B-BMF :** la version (Default) a commencé avec une RMSE de 1.48 et +- 0.17 de marge d'erreur pour un bruit égale à 0 ensuite 1.15 avec +- 0.14 de marge d'erreur et pour un bruit égale à 1 et finalement une RMSE de 1.16 avec +- 0.14 de marge d'erreur pour un bruit égale à 2. La variante (Bagging) a légèrement amélioré les performances par rapport à la version (Default) avec une différence de RMSE de 0.02. Tandis que le BaggingV2 a mieux géré le bruit, réduisant les RMSE de 1.13 à 1.00 ensuite à 1.11.

3.4.2.2 Évaluation avec 10% de données d'entraînement

Script d'exécution Python :

```
python evalRec.py --algorithm svd knn nmf base bmf --oracle ml-1m
--training_percentage 10 --noise_level 0 1 2 --seed 42
```

Algorithmes	**SVD**			**KNN**			**NMF**			**BASE**			**B-BMF**		
Versions	D	B	B2	D	B	B2	D	B	B2	D	B	B2	D	B	B2
0	0.94	0.94	0.95	0.96	0.96	0.97	1.02	0.96	0.99	0.94	0.94	0.95	1.55	1.41	1.30
1	0.97	0.96	0.99	1.01	1.00	1.01	1.10	1.00	1.04	0.97	0.97	0.98	1.29	1.25	1.23
2	1.03	1.02	1.06	1.11	1.10	1.12	1.27	1.10	1.15	1.02	1.02	1.04	1.43	1.33	1.30

Table 3.14 ML-1M - RMSE VS Niveaux de Bruit (10% de données d'entraînement)

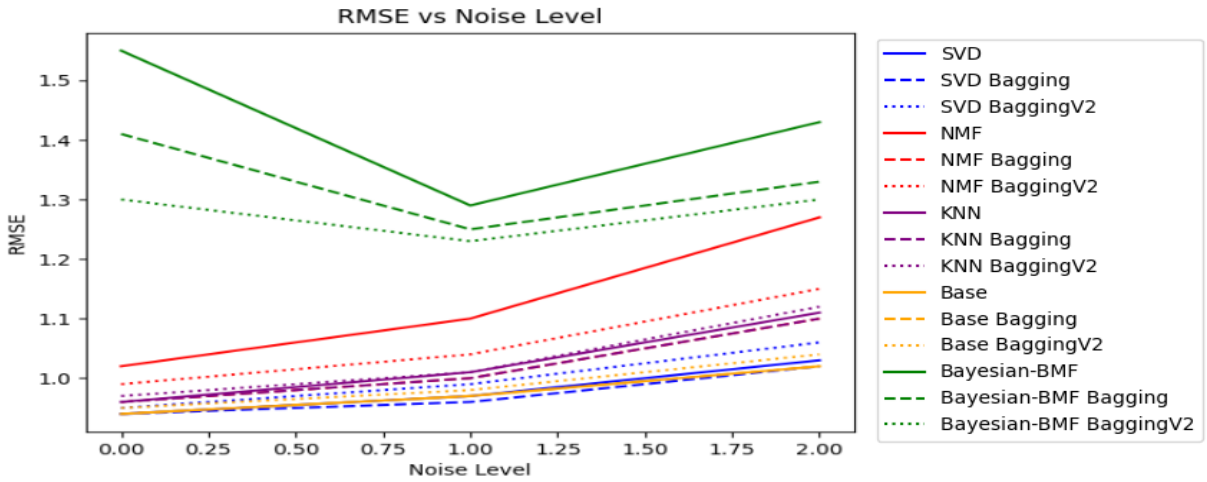


Figure 3.16 ML-1M - RMSE VS Niveaux de bruit (10% de données d'entraînement)

Observations :

- **BASE & SVD :** Comme dans le cas précèdent, SVD a encore le mieux performé pour tous les niveaux de bruits avec ses versions (Bagging) et (Default). BASE avait des résultats similaires à SVD pour tous les niveaux de Bruit avec ses versions (Bagging) et (Default).
- **KNN :** Un comportement similaire au cas précèdent. KNN était classé troisième, après SVD et BASE avec des performances très proches entre ses trois versions (Default) (Bagging) et (BaggingV2).
- **NMF :** Performances faibles par rapport au cas précèdent de 90% de données d'entraînement. NMF avec ses versions (Default) et (BaggingV2), avait des résultats moins bons que SVD, BASE et KNN. sa variante (Bagging) a significativement amélioré ses performances finissant avec des performances similaires à KNN.
- **B-BMF :** Des performances moins bonnes que les autres algorithmes et un comportement similaire avec 90% de données d'entraînement. Sauf qu'avec 10% de données d'entraînement, la version BaggingV2 de B-BMF avait des résultats proches de la version (Default) de NMF pour le niveau de bruit égale à 2.

3.5 GPT

Dans le cadre de la recherche, nous voulions évaluer les points forts et les lacunes de ChatGPT dans le domaine des systèmes de recommandation. Dans le cadre de la recherche, nous voulions évaluer les points forts et les lacunes de ChatGPT dans le domaine des systèmes de recommandation. Afin de limiter les coûts d'utilisation et en raison de la dégradation des performances de GPT avec l'augmentation de la taille des jeux de données, nous avons choisi d'effectuer cette étude préliminaire en utilisant GPT-3.5-turbo-1106 avec des ensembles de données de petite taille, pour déterminer si GPT est capable d'arriver à fournir de bonnes performances en tant qu'un modèle de recommandation ou pas. Pour ce faire, nous avons utilisé des oracles O-NMF avec un rang de 2 et des Oracles O-ADA Aléatoire d'une probabilité de feuille de 30%. Les ensembles de données comprenaient 4 utilisateurs et 20 films, ce qui permettait de rester dans les limites de capacité de GPT.

Nous avons fixé le niveau de bruit à 0 afin de simplifier l'analyse et de nous concentrer sur l'efficacité de l'algorithme. Les pourcentages d'entraînement ont été variés, avec des tests effectués à 50% et 90%. Chaque configuration de test a été répétée 11 fois, ce qui nous a permis de calculer une moyenne des résultats obtenus et d'assurer ainsi une meilleure fiabilité de nos conclusions. Ces tests ont permis d'obtenir des informations sur la performance de GPT dans des environnements contrôlés et de petite échelle, fournissant une base pour des comparaisons avec les autres algorithmes de recommandation ainsi qu'un algorithme MEAN qui prédit les notes simplement en calculant les moyennes des notes des utilisateurs et des films.

3.5.1 GPT VS O-NMF

Script d'exécution Python :

```
python evalRec.py --algorithm svd knn nmf base bmf gpt mean --oracle nmf
--num_users 4 --num_items 20 --rank 2 --training_percentage 50 90
--noise_level 0 --seed 42 --num_tests 11
```

Algorithmes	GPT	Base	SVD			KNN			NMF			BASE			B-BMF		
Versions	D	D	D	B	B2	D	B	B2	D	B	B2	D	B	B2	D	B	B2
50%	0.72	0.62	0.43	0.43	0.41	0.35	0.36	0.37	0.28	0.28	0.39	0.45	0.45	0.44	1.33	1.60	2.64
90%	0.70	0.49	0.38	0.39	0.36	0.23	0.24	0.24	0.12	0.10	0.18	0.43	0.44	0.40	0.84	0.58	1.88

Table 3.15 O-NMF - GPT

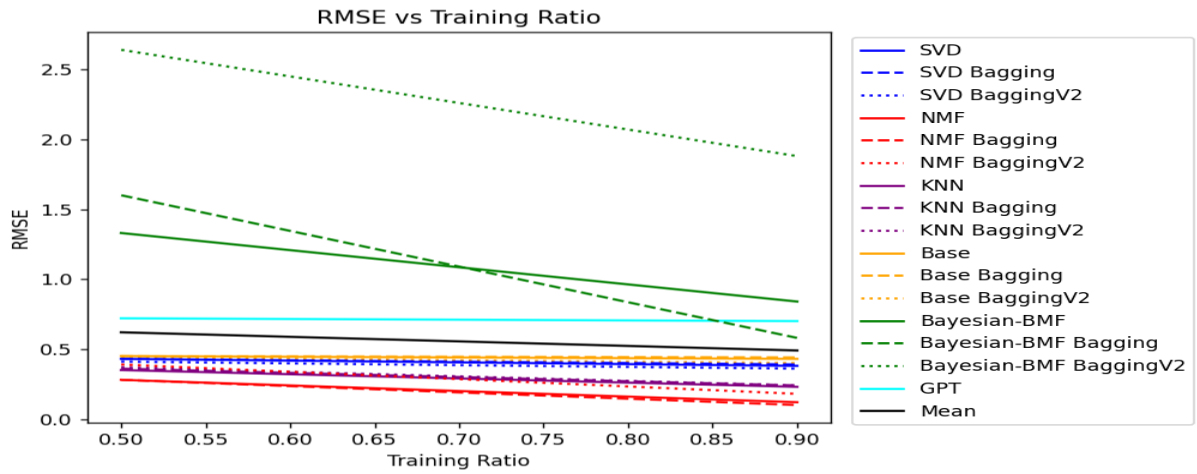


Figure 3.17 O-NMF - RMSE VS Pourcentages d'entraînement

Observations :

- Bayesian-BMF a eu des performances inférieures que le reste des algorithmes, sauf pour sa version bagging, qui a amélioré ses performances pour le pourcentage d'entraînement 90%, tandis que les résultats de BaggingV2 étaient moins bons que la version de base et que le reste des algorithmes.
- l'algorithme GPT a globalement mieux performé que le Bayesian BMF et il n'était pas très loin du reste des algorithmes pour ce test.
- l'algorithme Moyenne, avait des résultats RMSE légèrement inférieures que celles de GPT.
- SVD et BaseLine ont maintenu des performances stables avec des RMSE autour de 0.39 pour un ratio d'entraînement de 90%.
- les algorithmes KNN et NMF ont mieux performé, particulièrement avec des ratios d'entraînement de 90% avec des RMSE entre 0.35 et 0.10 pour NMF Bagging.

3.5.2 GPT VS O-ADA

```
python evalRec.py --algorithm svd knn nmf base bmf gpt mean --oracle random_trees
--num_users 4 --num_items 20 --leaf_probability 30 --training_percentage 50 90
--noise_level 0 --seed 42 --num_tests 11
```

Algorithmes	GPT	Base	SVD			KNN			NMF			BASE			B-BMF		
Versions	D	D	D	B	B2	D	B	B2	D	B	B2	D	B	B2	D	B	B2
50%	1.58	1.86	1.14	1.15	1.19	1.46	1.47	1.38	1.44	1.41	1.42	1.22	1.22	1.25	2.68	2.33	2.91
90%	1.45	1.73	1.15	1.15	1.19	1.31	1.32	1.36	1.19	1.17	1.30	1.20	1.20	1.24	1.74	1.78	1.83

Table 3.16 O-ADA - GPT

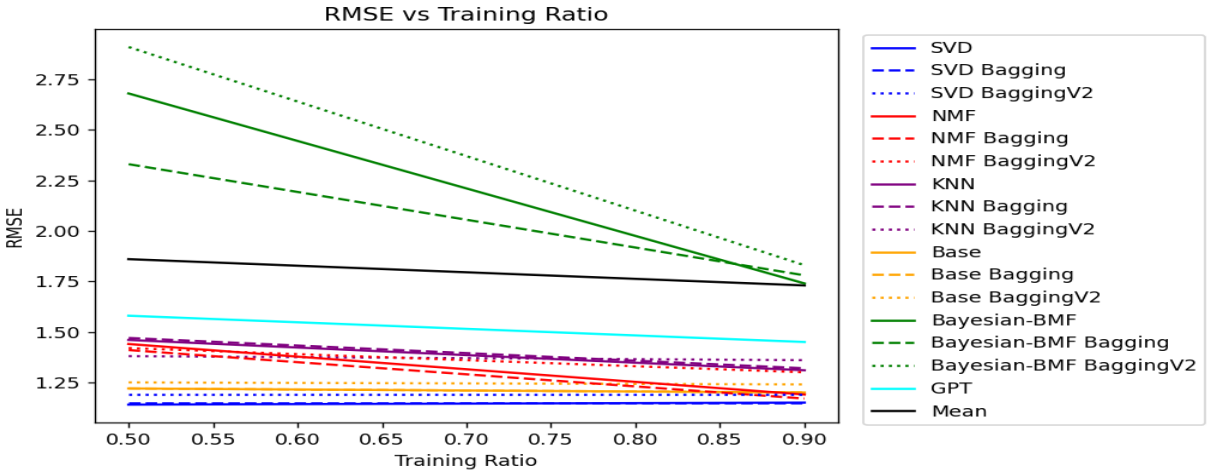


Figure 3.18 O-ADA - RMSE VS Pourcentages d'entraînement

Observations :

- l'algorithme BaseLine montre des performances stables à travers ses différentes versions, avec des RMSE variant peu entre le modèle par défaut et les versions Bagging et BaggingV2.
- SVD offre les meilleurs résultats en termes de RMSE, avec des valeurs autour de 1.15 à 1.19. Les baggings n'ont pas apporté d'améliorations.
- NMF présente des performances pas loin de celles de SVD avec le pourcentage d'entraînement 90%, mais les versions Bagging et BaggingV2 n'apportent pas de gains significatifs en termes de RMSE
- KNN a également eu des valeurs stables pour 50 et 90% de données d'entraînement.
- Par contre, dans ce cas, GPT a montré des performances meilleures que celles de l'algorithme MEAN et des résultats très proches de KNN.
- Bayesian-BMF a obtenu les pires résultats, bien que la version Bagging ait légèrement amélioré les performances pour un ratio d'entraînement de 50%.

CHAPITRE 4
DISCUSSION

Dans ce chapitre, nous allons discuter, en détails, les résultats que nous avons obtenus lors de nos tests sur les différents algorithmes de recommandation sur des différents types d'ensembles de données.

4.1 Interprétation des Résultats

Comme attendu, nous constatons que la performance des algorithmes s'améliore avec des ratios d'entraînement plus élevés. En d'autres termes, plus la proportion de données utilisées pour entraîner le modèle est grande, plus le modèle tend à produire des meilleures prédictions. Cependant, nous avons observé que cette amélioration a tendance à ne plus s'améliorer au-delà d'un certain point. Cela signifie que, bien que l'augmentation du ratio d'entraînement améliore initialement les performances, les gains supplémentaires deviennent de moins en moins significatifs en utilisant plus de données d'entraînement.

Nous avons également constaté que, certains algorithmes peuvent se révéler plus performants avec peu de données et moins efficaces avec un volume de données plus important comparativement a d'autre algorithmes. Par exemple pour ML-1M et ML-100K, on trouve que l'algorithme SVD avait les meilleurs RMSE quelque soit la quantité des données d'entraînement, cependant, l'algorithme BaseLine avait des meilleurs résultats par rapport au reste des algorithmes et presque exactement les mêmes performances que SVD en termes de RMSE pour 1% à 15% de données d'entraînement. Cela prouve que pour le cas de Movielens, BaseLine et SVD sont meilleurs que B-BMF, NMF et KNN pour des petites quantités de données d'entraînement.

Nous avons également observé un comportement similaire en ce qui concerne les Arbres de Décision Aléatoires. L'algorithme SVD a montré les meilleures performances pour des grandes quantités de données suivi par KNN et NMF qui a montré des résultats très proches les unes des autres, ensuite BASE et finalement B-BMF. Cependant, NMF s'est distingué en termes de performances de manière significativement plus efficace pour 1 à 35% de données d'entraînement, avec une RMSE allant d'environ 1.03 jusqu'à 0.53, tandis que les algorithmes SVD, BASE et KNN ont produit des RMSE allant d'environ 1.15 à 0.65 pour SVD, de 1.23 à 0.85 pour BASE et finalement de 1.30 à 0.68 pour KNN. Cela prouve que NMF est meilleur que les autres algorithmes dans le cas des Arbres de Décision Aléatoires en termes de performance avec peu de données.

Pour l'oracle O-NMF, nous avons observé un comportement similaire. KNN a obtenu les meilleurs résultats à partir de 20% de données d'entraînement et jusqu'à la fin de l'expérience allant de 0.07 à 0.02. NMF, SVD, et BASE ont montré des performances équivalentes avec de grandes quantités de données d'entraînement. Cependant, au début de l'expérience, NMF a surpassé de manière significative tous les autres algorithmes, pour des pourcentages d'entraînement inférieurs à 10%, affichant une RMSE stables de 0.10 de 5% jusqu'à la fin de l'expérience. SVD et BASE ont montré des résultats légèrement meilleurs que KNN avec moins de 4% de données, et SVD a continué à obtenir de meilleurs résultats que KNN jusqu'à 10%. Cela prouve que NMF est meilleur que les autres algorithme dans le cas des oracles O-NMF en termes de performances avec peu de données.

Oracle / % Entrainement	O-ADA	O-NMF	ML-100K	ML-1M
1% - 15%	NMF	NMF	BASE & SVD	SVD
16% - 40%	NMF	KNN	BASE & SVD	SVD
> 40%	SVD	KNN	SVD	SVD

Figure 4.1 Méthodes les plus performantes face aux Pourcentages d'entraînements - Tableau Récapitulatif

Concernant les niveaux de rang, on observe que NMF maintient des performances stables avec une RMSE égale à 0.07 quelles que soient les valeurs de rang. KNN, en revanche, obtient les meilleurs résultats avec de petites valeurs de rang (de 1 à 5) avec une RMSE égale à 0.02. Tandis que, SVD et BASE nécessitent des valeurs de rang supérieures à 5 pour atteindre des performances proches à celles de KNN. Cela montre que KNN est meilleur que les autres algorithme dans le cas des oracles O-NMF en termes de performances avec des rangs réduits.

4.2 Gestion du bruit :

L'ajout du bruit a montré un impact sur les performances des différents algorithmes testés. En nous basant sur les résultats obtenus à différents niveaux de bruit et différents types d'ensembles de données, nous analyserons comment chaque algorithme réagit aux perturbations et verrons quel algorithme performe le mieux avec les versions (Default) de chaque algorithme.

MovieLens :

En examinant la gestion du bruit des différents algorithmes sur les ensembles de données ML-100K et ML-1M, nous observons plusieurs tendances communes. Pour les deux ensembles de données :

- **Pas de Bruit :** Pour ML-100K, avec 90% de données d'entraînement, les performances des algorithmes sont très proches les unes des autres : SVD et KNN affichent une RMSE de 0,93. BASE et NMF présentent des RMSE de 0,94 et 0,95 respectivement, tandis que B-BMF avait des grandes valeurs RMSE de 1.52 pour sa version (Default). Par contre, avec 10% de données d'entraînement, on remarque que les trois versions (Default) (Bagging) et (BaggingV2) de SVD et BASE avaient des meilleures performances que tous les autres algorithmes, avec des valeurs RMSE exactement les mêmes de 1.00, 1.00 et 1.01, respectivement.

  Pour ML-1M, l'algorithme SVD se distingue avec une RMSE de 0.86, suivi par KNN et BASE avec des RMSE de 0.90 et NMF avec une RMSE légèrement plus élevée à 0.92. B-BMF avait une autre fois des performances très faibles avec une RMSE de 1.48 pour sa version (Default). Avec 10% de données d'entraînement et comme dans le cas de ML-100k, L'algorithme BASE à rejoint SVD dans la première position avec une RMSE de 0.94.

- **Bruit = 1 :** Pour ML-100K, avec 90% de données d'entraînement, les performances des algorithmes sont, également, très proches les unes des autres, comme pour le cas précèdent : SVD et BASE montrent une RMSE de 0.96, tandis que KNN et NMF présentent des RMSE de 0.97 et 0.99 respectivement. Tandis que B-BMF avait des performances meilleures que celles avec un bruit égale à 0 avec une RMSE de 1.19. Avec 10% de données d'entraînement, on constate que BASE et SVD on encore très bien performé, avec des valeurs RMSE très similaires de 1.03.

  Pour ML-1M, SVD et BASE montrent une RMSE de 0.92, suivi par NMF et KNN avec des RMSE de 0.93 et 0.94 respectivement. B-BMF avait des performances meilleures que celles avec un bruit égale à 0 avec une RMSE de 1.15. Comme avec ML-100K, avec 10% de données d'entraînement, les deux al-

gorithmes SVD et BASE avaient les meilleures performances avec des RMSE de 0.97.

- **Bruit = 2 :** Pour ML-100K, BASE se distingue avec la meilleure performance, avec 90% de données d'entraînement, affichant une RMSE de 1.01 suivi par KNN avec une RMSE de 1.05, SVD avec 1.07, NMF avec 1.08, et enfin B-BMF avec une RMSE de 1.24 et comme dans les deux cas précédents, avec 10% de données d'entraînement, BASE et SVD continuent à avoir les meilleurs performances avec des RMSE de 1.09.

  Pour ML-1M, BASE maintient une meilleure performance avec une RMSE de 0.97, légèrement meilleure que celle de NMF à 0.98. KNN et SVD suivent avec des RMSE de 1.03 et 1.04 respectivement, tandis que B-BMF enregistre la plus élevée avec une RMSE de 1.15. Comme dans les cas précédents, encore une fois NMF et SVD obtiennent les meilleurs résultats avec des RMSE de 1.02. Cette expérience prouve leur excellente gestion de bruit avec peu de données, par rapport aux autres algorithmes.

Oracle O-NMF :

- **BASE :** Avec 90% de données d'entraînement, BASE affiche des RMSE de 0.10, 0.15 et 0.29 pour les niveaux de bruit 0, 1 et 2, montrant une robustesse notable face au bruit, commençant l'expérience à la deuxième position ensuite à la première position pour les niveaux de bruit 1 et 2. Par contre, avec 10% de données d'entraînement, BASE commence l'expérience à la 4ème position avec une RMSE de 0.22 ensuite, pour un niveau de bruit égale à 1, BASE obtient la deuxième position avec une RMSE de 0.30. Et pour le niveau de bruit égale à 2, BASE obtient la première position avec une RMSE égale à 0.44.
- **NMF :** Avec 90% de données d'entraînement, NMF montre des RMSE de 0.10, 0.17 et 0.35, démontre également une gestion efficace du bruit. NMF obtient la deuxième position pour tous les niveaux de bruit. Avec 10% de données d'entraînement, NMF commence, encore une fois, l'expérience à la première position avec une RMSE de 0.10. Mais pour les deux derniers niveaux de bruit, NMF finit l'expérience à l'avant dernière place avec des RMSE, respectivement, de 0.51 et 0.89.
- **KNN :** Commence avec une excellente RMSE de 0.02, mais malgré une très bonne performance initiale, il montre une dégradation plus prononcée avec une RMSE de 0.23 pour un bruit égale à 1 ensuite une RMSE de 0.42 au niveau de bruit 2. KNN a commencé à la première position, ensuite la 4ème et finissant l'expérience à la 3ème position. KNN produit une performance similaire, avec 10% de données d'entraînement, il a obtenu des RMSE de 0.11 , 0.33 et 0.52 pour les trois niveaux de bruits respectivement, commençant à la deuxième place et finissant l'expérience à la 3ème position pour les niveaux de bruits 1 et 2.

— **SVD :** Présente des RMSE de 0.09, 0.19 et 0.50, avec une dégradation plus marquée au niveau de bruit 2, indiquant une moindre robustesse et finissant avec les pires résultats parmi NMF, KNN, et BASE pour le niveau de bruit égale à 2. Par contre, Avec 10% de données d'entraînement, SVD commence l'expérience à la troisième position avec une RMSE de 0.13 après NMF et KNN mais pour les deux niveaux de bruits 1 et 2, SVD a fini eu la première place avec des RMSE de 0.27 et 0.44 respectivement.

— **B-BMF :** Produit des RMSE plus élevés de 0.66, 0.70 et 0.68, indiquant une mauvaise performance initiale pour sa version (Default), mais une bonne gestion de bruit pour les niveaux 0, 1 et 2. Avec 10% de données d'entraînement, B-BMF a obtenu des RMSE de 0.74, 0.81 et 1.02, finissant l'expérience à la dernière position.

Oracle O-ADA :

— **SVD :** Pour 90% de données d'entraînement, Affiche les meilleures performances globales pour tous les niveaux de bruit, commençant avec une RMSE de 0,27 pour le bruit = 0, puis 0,45 pour le bruit = 1, et se terminant à 0,78 pour le bruit = 2. Malgré une dégradation notable, SVD reste le meilleur algorithme en termes de performances globales.Par contre, Avec 10% de données d'entraînement, SVD commence l'expérience à la troisème position après NMF et KNN avec une RMSE de 0.87 ensuite il réussit à obtenir la première place avec une RMSE de 0.94, avant la version (Default) de NMF, finalement, pour un bruit égale à 2 il obtient une RMSE de 1.06, finissant, encore une fois, à la première position.

— **KNN & NMF :** Présentent des performances similaires, avec des RMSE débutant respectivement à 0.48, 0.60, et 0.84 pour KNN et 0.46, 0.61, et 0.86 pour NMF. Ils gèrent le bruit de manière plus efficace que SVD pour un pourcentage d'entraînement de 90%, mais leurs valeurs de RMSE sont inférieures. Les performances de NMF et KNN étaient, également, similaires pour un pourcentage d'entraînement de 10%. Ils ont commencé l'expérience, dans la première et deuxième position, respectivement, avec des RMSE de 0.81 et 0.85 pour un bruit égale 0. Ensuite avec des RMSE de 0.97 et 0.98 pour un bruit égale à 2. Finalement, pour le niveau 2 de bruit, Ils ont fini l'expérience des RMSE de 1.15 et 1.26, montrant une différence notable au niveau des RMSE pour un bruit de niveau 2.

— **BASE :** Commence avec une RMSE de 0.83, évolue à 0.88, et atteint 0.98. Bien que BASE offre une excellente gestion du bruit, malgré que ses performances globales sont inférieures comparées aux autres algorithmes. Ensuite pour 10% de données d'entraînement, BASE continue sa bonne gestion de bruit avec des RMSE de 0.99 , 1.06 et 1.16

— **BMF :** Quant a lui, affiche des RMSE plus élevées de 1.74, 1.31, et 1.40, pour 90% de données d'entraî-

nement, et des RMSE de 1.73, 1.51 et 1.65 avec 10% de données d'entraînement. Il fini l'expérience à la dernière position.

Nous pouvons conclure que BASE et SVD se distinguent comme les meilleurs algorithmes pour la gestion du bruit, montrant une robustesse à travers les différents ensembles de données. Ils obtiennent les meilleurs résultats pour la plupart des cas, notamment avec des grands ou petits pourcentages de données d'entraînement, et ils maintiennent une bonne performance malgré les variations de bruit. Toutefois, il est important de noter qu'aucun algorithme ne parvient à obtenir systématiquement les meilleures performances. Le classement des algorithmes varie en fonction de l'ensemble de données utilisé.

Les tableaux ci-dessous résument les dernières expériences.

Oracle / Bruit	O-ADA	O-NMF	ML-100K	ML-1M
Bruit = 0	SVD	KNN	SVD&KNN	SVD
Bruit = 1	SVD	BASE	SVD&BASE	SVD&BASE
Bruit = 2	SVD	BASE	BASE	BASE

Figure 4.2 Gestion de bruit (90% de données d'entraînement) - Tableau Récapitulatif

Oracle / Bruit	O-ADA	O-NMF	ML-100K	ML-1M
Bruit = 0	NMF	NMF	SVD & BASE	SVD & BASE
Bruit = 1	SVD	SVD	SVD & BASE	SVD & BASE
Bruit = 2	SVD	SVD & BASE	SVD & BASE	SVD & BASE

Figure 4.3 Gestion de bruit (10% de données d'entraînement) - Tableau Récapitulatif

4.3 Efficacité des versions de Bagging :

On remarque que le (BaggingV2) était efficace avec l'algorithme SVD pour O-ADA augmentant les RMSE avec une différence d'environ 0.07 par rapport aux approches (Bagging) et (Default) pour les différents

pourcentages d'entraînement et les différentes probabilités de feuilles. Cependant, Le reste des résultats de nos tests montrent que les deux approches (Bagging) et (BaggingV2), n'ont apporté aucune amélioration significative des performances dans la plupart des expériences réalisées en dehors de la gestion du bruit. En effet, que ce soit pour les performances des algorithmes avec différents tailles de données d'entraînements ou différents niveaux de rang pour l'Oracle NMF ou différentes probabilités de feuilles pour O-ADA, les versions de bagging n'ont pas démontré de bénéfices notables sauf pour quelques situations.

Pour Bayesian-BMF, par exemple, les versions de bagging ont montré des améliorations notables dans les contextes cités ci-dessous :

— **MovieLens :** La version BaggingV2 de l'algorithme B-BMF, a considérablement optimisé les résultats en réduisant le RMSE avec une différence de 0.30 (de 1.50 à 1.15) par rapport à la version (Default) pour Movielens-1M, et avec une différence de 0.29 (de 1.51 à 1.21) par rapport à la version (Default) pour ML-100K. Cette amélioration significative démontre l'efficacité de baggingv2 sur l'algorithme B-BMF sur MovieLens. Tandis que la version (Bagging) n'a pas apporté significatives.

— **Arbres de Décision Aléatoires :** Les versions de bagging ont également eu un impact positif. Nous avons observé que la première variante (Bagging) améliorait légèrement les performances globales. En revanche, (BaggingV2) s'est révélé plus efficace, offrant une amélioration avec une différence de 0.34 (de 1.47 à 1.13) en termes de RMSE pour les probabilités de feuilles par rapport à la version (Default) Et une différence de 0.32 (de 1.59 à 1.27) pour les pourcentages d'entraînement par rapport à la version (Default).

Cependant, lorsque le niveau de bruit augmente, les techniques de bagging montrent leur potentiel en offrant des améliorations remarquables dans certains cas, renforçant ainsi la robustesse de quelques algorithmes contre les perturbations :

— **SVD :** L'utilisation de (Bagging) entraîne une amélioration significative des résultats à travers tous les niveaux de bruit pour tous les ensembles de données. (BaggingV2), bien que bénéfique, améliore les performances dans une moindre mesure comparé au (Bagging). Cela indique que (Bagging) est particulièrement efficace pour renforcer les performances de SVD dans des conditions bruyantes.
— **KNN & NMF :** Concernant KNN et NMF, (Bagging) a démontré une légère amélioration des résul-

tats, tandis que (BaggingV2) n'a eu aucun effet positif. Cela suggère que (Bagging) est plus adapté pour améliorer les performances de KNN et NMF, tandis que (BaggingV2) n'apporte pas d'avantages supplémentaires dans ce cas.

— **BASE :** En ce qui concerne BASE, ni (Bagging) ni (BaggingV2) n'ont eu un impact particulier sur les performances globales. Cependant, il est à noter que (BaggingV2) semble avoir légèrement détérioré les résultats par rapport à (Bagging) mais globalement, les deux versions n'ont pas entraîné de changements significatifs dans la gestion du bruit.

— **B-BMF :** , (Bagging) a apporté de bonnes amélioration des performances, mais c'est (BaggingV2) qui a montré une amélioration significative dans tous les cas. Ce résultat indique que (BaggingV2) est particulièrement efficace pour améliorer les performances de B-BMF.

Ci-dessous deux tableaux récapitulatifs des observations mentionnées ci-dessus :

Oracle / Bruit	O-ADA	O-NMF	ML-100K	ML-1M
Bruit = 0	BaggingV2(SVD)	Bagging(KNN)	Bagging(SVD)	Bagging(SVD)
Bruit = 1	Bagging(SVD)	Bagging(SVD &KNN&BASE)	Bagging(SVD &NMF)	Bagging(SVD)
Bruit = 2	Bagging(SVD)	Bagging(BASE)	Bagging(SVD &BASE)	Bagging(NMF)

Figure 4.4 Bagging face au Bruit (90% de données d'entraînement) - Tableau Récapitulatif

Oracle / Bruit	O-ADA	O-NMF	ML-100K	ML-1M
Bruit = 0	Bagging(NMF)	BaggingV2(NMF)	SVD & BASE & Bagging(SVD &BASE)	SVD & BASE & Bagging(SVD &BASE)
Bruit = 1	Bagging(NMF)	Bagging(SVD)	Bagging(SVD)	Bagging(SVD)
Bruit = 2	Bagging(NMF)	Bagging(SVD)	3 versions(BASE) Bagging(SVD)	BASE & Bagging(SVD &BASE)

Figure 4.5 Bagging face au Bruit (10% de données d'entraînement) - Tableau Récapitulatif

4.4 O-NMF & O-ADA VS ML-100K & ML-1M :

Lors de la comparaison des oracles O-NMF et O-ADA avec les jeux de données MovieLens, nous avons observé des tendances distinctes en fonction des pourcentages d'entraînement et des niveaux de bruit.
Pour les différents pourcentages d'entraînement, les RMSE obtenus variaient, mais l'ordre des algorithmes restait relativement constant à travers toutes les expériences : SVD se classait systématiquement en tête (sauf dans le cas de O-NMF ou KNN avait les meilleures résultats avec un rang égale à 2), suivi de KNN, BASE et NMF avec des différences minimales, tandis que BMF se retrouvait systématiquement en dernière position. Concernant la gestion du bruit, les quatre ensembles de données ont montré des comportements similaires. Pour SVD, la performance était initialement bonne en l'absence de bruit, mais se détériorait fortement avec l'ajout de bruit. Les versions de bagging se sont révélées efficaces pour atténuer cet effet. Et l'algorithme BaseLine avait tout le temps les meilleures performances à travers tous les ensembles de données en termes de gestion de bruit, pas en termes de RMSE.
Nous avons également remarqué que les valeurs RMSE obtenues avec O-NMF et O-ADA étaient inférieures à celles observées sur MovieLens, cela est attribué au fait que le jeu de données MovieLens présente une complexité plus importante. Mais, avec O-NMF, un niveau de bruit inférieur à 2 n'était pas suffisant pour obtenir une RMSE comparable à celui observé dans les données de MovieLens. En revanche, pour O-ADA, un niveau de bruit égale à 2 était suffisant pour obtenir des résultats proches à ceux de Movielens.

4.5 Résultats de GPT :

Finalement, Nous avons évaluer le modèle GPT dans le cadre des systèmes de recommandation. Nos tests ont été dirigés vers des ensembles de données de petite taille, en utilisant l'Oracle O-NMF et l'Oracle O-ADA et deux pourcentages d'entraînement : 50% et 90%. Chaque configuration a été testée 11 fois, permettant de calculer une moyenne des résultats pour une meilleure fiabilité. Nous avons constaté que GPT a mieux performé que MEAN et Bayesian NMF pour l'oracle des Arbres de Décision Aléatoires, mais a moins bien performé sur l'oracle de factorisation matricielle O-NMF ce qui confirme que les oracles jouent un rôle crucial dans les performances des algorithmes de recommandation.

Pour conclure, nos tests ont révélé que, bien que l'augmentation du ratio d'entraînement et l'utilisation de techniques de bagging puissent améliorer les performances des algorithmes de recommandation, ces améliorations sont variables et dépendent fortement de l'algorithme et du jeu de données.

4.6 Limitations

Les expérimentations présentées dans ce document ont évidemment révélé quelques limitations. Tout d'abord, l'utilisation de GPT présente certaines limitations spécifiques. L'un des principaux défis rencontrés est la sensibilité élevée de GPT aux prompts et aux questions que nous devons poser. Les performances de l'algorithme peuvent varier considérablement en fonction de la formulation des requêtes, ce qui peut introduire une variabilité dans les résultats qui complique l'évaluation et la comparaison avec d'autres algorithmes. De plus, Les modèles GPT nécessitent des ressources de calcul importantes, ce qui peut rendre leur utilisation coûteuse pour des grandes bases de données. Cette limitation est due aux exigences élevées en termes de mémoire et de calcul nécessaires pour traiter des ensembles de données volumineux. Pour cette raison, nos tests ont été limités à des ensembles de données de petite taille.

Ensuite, la gestion du bruit dans les données reste un problème significatif. Bien que les techniques de bagging aient montré une certaine efficacité pour atténuer les effets du bruit, elles ne parviennent pas à éliminer complètement les impacts négatifs du bruit. Les résultats obtenus avec les données bruitées ne sont pas les mêmes que ceux obtenus avec les données originales non bruitées pour les algorithmes testés.

Un autre point à considérer est le temps d'exécution des techniques de bagging. Bien que ces techniques puissent améliorer les performances des algorithmes, elles nécessitent généralement plus de temps pour l'exécution des tâches. Cela peut poser des problèmes dans des environnements où les ressources et le temps sont limités.

CONCLUSION

Ce document a permis de réaliser une étude approfondie et comparative de divers algorithmes de recommandation, ce qui nous a offert l'opportunité d'atteindre nos objectifs, et de répondre à la problématique abordée dans ce travail de recherche :

La problématique principale abordée dans ce mémoire était de trouver une méthode pour évaluer des algorithmes de recommandation, en tenant compte des limitations des ensembles de données réels, souvent difficiles d'accès ou biaisés. Cette difficulté à accéder à des jeux de données de qualité et la présence de biais, pose un réel défi pour une évaluation précise et objective des algorithmes de recommandation. Pour répondre à ce défi, Nous avons développé un outil qui permet de contrôler précisément les caractéristiques des données à générer :

- Les Dimensions du jeu de données (le nombre d'utilisateurs et le nombre de films).
- Les niveaux de bruits.

L'outil, que nous avons implémenté, permet évaluer un certain nombre d'algorithmes de recommandation, notamment GPT, MEAN, KNN, SVD, NMF, BASE et Bayesian-BMF sur différents types d'ensembles de données. Les évaluations ont été réalisées sur quatre types de jeux de données différents :
1°) - Les oracles basés sur les arbres de décision aléatoires (O-ADA).
- Les oracles basés sur la factorisation de matrice non-négative (O-NMF)
2°) Deux jeux de données de référence bien connus, MovieLens ML-100K et ML-1M.

L'objectif de la génération des deux oracles est de simuler différents types de structures dans les données et évaluer ainsi des aspects variés des modèles de recommandation. L'oracle basé sur les arbres de décision aléatoires (O-ADA) génère des données avec des relations complexes entre les caractéristiques, tandis que l'oracle basé sur la factorisation de matrice non-négative (O-NMF) crée des données qui permettent de tester comment les modèles capturent les relations linéaires et latentes entre utilisateurs et items. Ces deux oracles permettent d'évaluer la robustesse des modèles face à différents types de structures de données.

Pour présenter les résultats de manière claire et détaillée, nous avons utilisé des tableaux et des graphiques, notamment des courbes montrant les RMSE des algorithmes en fonction des paramètres que nous avons

fait varier. Ces outils de visualisation ont facilité la compréhension et l'interprétation des performances des différents algorithmes.

Dans le cadre de ces évaluations, Nous avons également mis en œuvre deux techniques pour améliorer la performance des algorithmes : le Bagging classique et une variante nommée BaggingV2. Cette dernière, simule un algorithme de Boosting sans nécessiter de modifier les algorithmes pour qu'ils prennent en compte des données pondérées. Ces techniques ont été testées pour déterminer dans quelles situations elles montrent leur potentiel d'amélioration des performances.

Les expériences menées ont démontré que l'algorithme BaggingV2, que nous avons développé, améliore effectivement les performances des algorithmes de recommandation dans un certain nombre de cas. La technique classique de Bagging montre également une amélioration significative des résultats dans de nombreuses situations. Par ailleurs, nous avons testé ChatGPT en tant qu'un algorithme de recommandation. Dans ce cadre, GPT a montré qu'il pouvait réaliser des recommandations en utilisant uniquement des historiques réduits de notes. Bien qu'il soit moins performant que la plupart des algorithmes spécialisés, il parvient à dépasser l'algorithme Bayesian-BMF et l'algorithme MEAN en termes de performance, qui est un algorithme simple qui calcule les moyennes des notes.

Les résultats de nos expériences ont permis de conclure qu'il n'existe pas d'algorithme qui soit systématiquement le meilleur dans toutes les conditions. Les performances des algorithmes dépendent fortement du jeu de données et des paramètres spécifiques de l'évaluation. Par exemple, certains algorithmes peuvent performer mieux, avec peu de données ou dans des contextes avec un certain niveau de bruit, que d'autres algorithmes. De plus, nous avons constaté que, les intervalles des RMSE varient en fonction de la complexité du jeu de données. En conséquence, le choix de l'algorithme le plus adapté doit être fait en tenant compte des caractéristiques particulières du jeu de données et des conditions expérimentales.

Les oracles proposés O-ADA et O-NMF, permettent d'obtenir une évaluation plus ciblée des algorithmes de recommandation. Contrairement à un jeu de données comme MovieLens, ces oracles offrent un contrôle plus précis sur les caractéristiques sous-jacentes des données, comme les relations latentes et la distribution des notes. En combinant ces oracles, on obtient une évaluation plus complète et plus robuste des modèles, ce qui est difficile à obtenir avec un seul jeu de données. Les oracles peuvent être adaptés aux spécificités du problème de recommandation, en ajustant leurs paramètres pour simuler différents types de données.

Les résultats obtenus montrent que l'algorithme SVD performe mieux que NMF lorsque suffisamment de données sont disponibles, tandis que NMF prend l'avantage lorsque les données sont limitées. En ce qui concerne la factorisation de matrice booléenne (B-BMF), les résultats se sont révélés bien inférieurs aux algorithmes de recommandation classiques. Cependant, l'application des deux versions de Bagging que nous avons implémentées, a permis d'améliorer de manière significative ces résultats, démontrant une fois de plus l'efficacité de ces techniques dans certaines conditions.

Cet outil, nous a permis de tester et de comparer diverses approches de recommandation, de développer des techniques pour améliorer les performances des algorithmes de recommandation existants, et de comprendre les avantages et les limitations de chaque approche. Les résultats obtenus grâce à cette étude fournissent une base solide pour des recherches futures et des applications pratiques dans le domaine des systèmes de recommandation.

Vous pouvez accéder au code complet sur GitHub en suivant ce lien : Code GitHub

BIBLIOGRAPHIE

Abdulla, G. M. et Designs, M. (2017). Size recommendation system for fashion e-commerce. https://www.semanticscholar.org/paper/Size-Recommendation-System-for-Fashion-E-commerce-Abdulla-Designs/fa1d08c3fc643c0e41d3ed2e688bf9c9b389f066.

Alyari, F. et Jafari Navimipour, N. (2018). Recommender systems : A systematic review of the state of the art literature and suggestions for future research. *Kybernetes*, *47*(5), 985–1017. http://dx.doi.org/10.1108/K-06-2017-0196

Asoke Nath, Adityam Ghosh, A. M. (2018). Building a movie recommendation system using svd algorithm. *International Journal of Computer Sciences and Engineering*, *6*, 727–729. http://dx.doi.org/https://doi.org/10.26438/ijcse/v6i11.727729. Récupéré de https://www.ijcseonline.org/full_paper_view.php?paper_id=3234

Belletti, F., Lakshmanan, K., Krichene, W., Chen, Y.-F. et Anderson, J. (2019). Scalable realistic recommendation datasets through fractal expansions. Récupéré de https://arxiv.org/abs/1901.08910

Bennett, J. et Lanning, S. (2007). The netflix prize. Récupéré de https://api.semanticscholar.org/CorpusID:9528522

Bhardwaj, A. (2023). Movie recommendation system using svd (letterboxd). *IJARCCE*, *12*. http://dx.doi.org/10.17148/IJARCCE.2023.121013

Brunton, S. L. et Kutz, J. N. (2019). Singular value decomposition (svd). In *Data-Driven Science and Engineering : Machine Learning*, *Dynamical Systems*, *and Control* chapitre 1, 3–46. Cambridge University Press

Cami, B. R., Hassanpour, H. et Mashayekhi, H. (2017). A content-based movie recommender system based on temporal user preferences. Dans *2017 3rd Iranian Conference on Intelligent Systems and Signal Processing (ICSPIS)*, 121–125. http://dx.doi.org/10.1109/ICSPIS.2017.8311601

Dankar, F. K. et Ibrahim, M. (2021). Fake it till you make it : Guidelines for effective synthetic data generation. *Applied Sciences*, *11*(5). http://dx.doi.org/10.3390/app11052158. Récupéré de https://www.mdpi.com/2076-3417/11/5/2158

del Carmen Rodríguez-Hernández, M., Ilarri, S., Hermoso, R. et Trillo-Lado, R. (2017). Datagencars : A generator of synthetic data for the evaluation of context-aware recommendation systems. *Pervasive and Mobile Computing*, *38*, 516–541. Special Issue IEEE International Conference on Pervasive Computing and Communications (PerCom) 2016, http://dx.doi.org/https://doi.org/10.1016/j.pmcj.2016.09.020. Récupéré de https://www.sciencedirect.com/science/article/pii/S157411921630270X

Ekstrand, M. D., Chaney, A., Castells, P., Burke, R., Rohde, D. et Slokom, M. (2021). Simurec : Workshop on synthetic data and simulation methods for recommender systems research. Dans *Proceedings of the 15th ACM Conference on Recommender Systems*, RecSys '21, p. 803–805., New York, NY, USA.

Association for Computing Machinery. http://dx.doi.org/10.1145/3460231.3470938. Récupéré de https://doi.org/10.1145/3460231.3470938

Gillis, N. (2014). The why and how of nonnegative matrix factorization. Récupéré de https://arxiv.org/abs/1401.5226

Gonçalves, A., Ray, P., Soper, B., Stevens, J., Coyle, L. et Sales, A. (2020). Generation and evaluation of synthetic patient data. *BMC Medical Research Methodology*, *20*.

Green, D. et Bailey, S. (2024). Algorithms for non-negative matrix factorization on noisy data with negative values. Récupéré de https://arxiv.org/abs/2311.04855

Harper, F. M. et Konstan, J. A. (2015). The movielens datasets : History and context. *ACM Trans. Interact. Intell. Syst.*, *5*(4). http://dx.doi.org/10.1145/2827872. Récupéré de https://doi.org/10.1145/2827872

Hernandez, M., Epelde, G., Alberdi, A., Cilla, R. et Rankin, D. (2022). Synthetic data generation for tabular health records : A systematic review. *Neurocomputing*, *493*, 28–45. http://dx.doi.org/https://doi.org/10.1016/j.neucom.2022.04.053. Récupéré de https://www.sciencedirect.com/science/article/pii/S0925231222004349

Hug, N. (2020a). Surprise : A python library for recommender systems. *Journal of Open Source Software*, *5*(52), 2174. http://dx.doi.org/10.21105/joss.02174. Récupéré de https://doi.org/10.21105/joss.02174

Hug, N. (2020b). Surprise : A python library for recommender systems. *Journal of Open Source Software*, *5*(52), 2174. http://dx.doi.org/10.21105/joss.02174. Récupéré de https://doi.org/10.21105/joss.02174

Ignatov, D., Nenova, E., Konstantinov, A. et Konstantinova, N. (2014). Boolean matrix factorisation for collaborative filtering : An fca-based approach.

Jakomin, M., Curk, T. et Bosnić, Z. (2018). Generating inter-dependent data streams for recommender systems. *Simulation Modelling Practice and Theory*, *88*, 1–16. http://dx.doi.org/https://doi.org/10.1016/j.simpat.2018.07.013. Récupéré de https://www.sciencedirect.com/science/article/pii/S1569190X18301059

Jayalakshmi, S., Ganesh, N., Čep, R. et Senthil Murugan, J. (2022). Movie recommender systems : Concepts, methods, challenges, and future directions. *Sensors*, *22*(13). http://dx.doi.org/10.3390/s22134904. Récupéré de https://www.mdpi.com/1424-8220/22/13/4904

Lee, D. et Seung, H. S. (2000). Algorithms for non-negative matrix factorization. Dans T. Leen, T. Dietterich, et V. Tresp (dir.). *Advances in Neural Information Processing Systems*, volume 13. MIT Press. Récupéré de https://proceedings.neurips.cc/paper_files/paper/2000/file/f9d1152547c0bde01830b7e8bd60024c-Paper.pdf

Lesnikowski, A., de Souza Pereira Moreira, G., Rabhi, S. et Byleen-Higley, K. (2021). Synthetic data and simulators for recommendation systems : Current state and future directions. *CoRR*,

abs/2112.11022. Récupéré de https://arxiv.org/abs/2112.11022

Li, Y., Liu, K., Satapathy, R., Wang, S. et Cambria, E. (2023). Recent developments in recommender systems : A survey. Récupéré de https://arxiv.org/abs/2306.12680

Liu, J., Liu, C., Zhou, P., Lv, R., Zhou, K. et Zhang, Y. (2023). Is chatgpt a good recommender ? a preliminary study. Récupéré de https://arxiv.org/abs/2304.10149

MacKenzie, I., Meyer, C. et Noble, S. (2013). How retailers can keep up with consumers. McKinsey & Company. Récupéré de https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers

Neumann, D., Lutz, A., Müller, K. et Samek, W. (2023). A privacy preserving system for movie recommendations using federated learning. *ACM Trans. Recomm. Syst.* Just Accepted, http://dx.doi.org/10.1145/3634686. Récupéré de https://doi.org/10.1145/3634686

Nikolenko, S. I. (2019). Synthetic data for deep learning. Récupéré de https://arxiv.org/abs/1909.11512

Palma, D. D., Biancofiore, G. M., Anelli, V. W., Narducci, F., Noia, T. D. et Sciascio, E. D. (2024). Evaluating chatgpt as a recommender system : A rigorous approach. Récupéré de https://arxiv.org/abs/2309.03613

Popić, S., Pavković, B., Velikić, I. et Teslić, N. (2019). Data generators : a short survey of techniques and use cases with focus on testing. Dans *2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin)*, 189–194. http://dx.doi.org/10.1109/ICCE-Berlin47944.2019.8966202

Prakash, A., Boochoon, S., Brophy, M., Acuna, D., Cameracci, E., State, G., Shapira, O. et Birchfield, S. (2020). Structured domain randomization : Bridging the reality gap by context-aware synthetic data. Récupéré de https://arxiv.org/abs/1810.10093

Provalov, V., Stavinova, E. et Chunaev, P. (2021). Synevarec : A framework for evaluating recommender systems on synthetic data classes. Dans *2021 International Conference on Data Mining Workshops (ICDMW)*, 55–64. http://dx.doi.org/10.1109/ICDMW53433.2021.00014

Rahul, M., Kumar, V., Yadav, V. et Rishabh (2021). Movie recommender system using single value decomposition and k-means clustering. *IOP Conference Series : Materials Science and Engineering*, *1022*(1), 012100. http://dx.doi.org/10.1088/1757-899X/1022/1/012100. Récupéré de https://dx.doi.org/10.1088/1757-899X/1022/1/012100

Rendle, S., Zhang, L. et Koren, Y. (2019). On the difficulty of evaluating baselines : A study on recommender systems. Récupéré de https://arxiv.org/abs/1905.01395

Rohde, D., Bonner, S., Dunlop, T., Vasile, F. et Karatzoglou, A. (2018). Recogym : A reinforcement learning environment for the problem of product recommendation in online advertising. Récupéré de https://arxiv.org/abs/1808.00720

Rukat, T., Holmes, C. C., Titsias, M. K. et Yau, C. (2017). Bayesian boolean matrix factorisation. Récupéré de https://arxiv.org/abs/1702.06166

Saadati, M., Shihab, S. et Rahman, M. S. (2019). Movie recommender systems : Implementation and performance evaluation. Récupéré de https://arxiv.org/abs/1909.12749

Sankaranarayanan, S., Balaji, Y., Jain, A., Lim, S. N. et Chellappa, R. (2018). Learning from synthetic data : Addressing domain shift for semantic segmentation. Récupéré de https://arxiv.org/abs/1711.06969

Slokom, M. (2018). Comparing recommender systems using synthetic data. Dans *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys '18, p. 548–552., New York, NY, USA. Association for Computing Machinery. http://dx.doi.org/10.1145/3240323.3240325. Récupéré de https://doi.org/10.1145/3240323.3240325

Slokom, M., Larson, M. et Hanjalic, A. (2020). Partially synthetic data for recommender systems : Prediction performance and preference hiding. Récupéré de https://arxiv.org/abs/2008.03797

Stavinova, E., Grigorievskiy, A., Volodkevich, A., Chunaev, P., Bochenina, K. et Bugaychenko, D. (2022a). Synthetic data-based simulators for recommender systems : A survey. Récupéré de https://arxiv.org/abs/2206.11338

Stavinova, E., Gurov, A., Lysenko, A. et Chunaev, P. (2022b). Performance ranking of recommender systems on simulated data. *Procedia Computer Science*, *212*, 142–151. 11th International Young Scientist Conference on Computational Science, http://dx.doi.org/https://doi.org/10.1016/j.procs.2022.10.216. Récupéré de https://www.sciencedirect.com/science/article/pii/S187705092201688X

Sun, L. et Erath, A. (2015). A bayesian network approach for population synthesis. *Transportation Research Part C : Emerging Technologies*, *61*, 49–62. http://dx.doi.org/https://doi.org/10.1016/j.trc.2015.10.010. Récupéré de https://www.sciencedirect.com/science/article/pii/S0968090X15003599

Tian, M. et Ekstrand, M. D. (2020). Estimating error and bias in offline evaluation results. Dans *Proceedings of the 2020 Conference on Human Information Interaction and Retrieval*, CHIIR '20, p. 392–396., New York, NY, USA. Association for Computing Machinery. http://dx.doi.org/10.1145/3343413.3378004. Récupéré de https://doi.org/10.1145/3343413.3378004

Tobin, J., Biewald, L., Duan, R., Andrychowicz, M., Handa, A., Kumar, V., McGrew, B., Schneider, J., Welinder, P., Zaremba, W. et Abbeel, P. (2018). Domain randomization and generative models for robotic grasping. Récupéré de https://arxiv.org/abs/1710.06425

Tremblay, J., Prakash, A., Acuna, D., Brophy, M., Jampani, V., Anil, C., To, T., Cameracci, E., Boochoon, S. et Birchfield, S. (2018). Training deep networks with synthetic data : Bridging the reality gap by domain randomization. Récupéré de https://arxiv.org/abs/1804.06516

Wes McKinney (2010). Data Structures for Statistical Computing in Python. Dans Stéfan van der Walt et Jarrod Millman (dir.). *Proceedings of the 9th Python in Science Conference*, 56 – 61. http://dx.doi.org/10.25080/Majora-92bf1922-00a

Zhang, Y. (2022). An introduction to matrix factorization and factorization machines in recommendation system, and beyond. Récupéré de https://arxiv.org/abs/2203.11026

Zhang, Z. (2016). Introduction to machine learning : k-nearest neighbors. *Annals of Translational Medicine*, *4*(11). Récupéré de https://atm.amegroups.org/article/view/10170