Projet : Système de Recommandation

Modèle Hybride

Auteur : Iyed Mekki

November 17, 2025

Contents

Introduction

Les systèmes de recommandation jouent un rôle crucial dans la personnalisation des services numériques, permettant de filtrer efficacement l'immense quantité d'informations disponibles et de proposer aux utilisateurs des contenus pertinents. Ce projet s'inscrit dans le domaine de la recherche sur les systèmes de recommandation, en s'appuyant sur le dataset MovieLens 1M, qui fournit des évaluations explicites de films par des milliers d'utilisateurs.

L'objectif principal de ce projet est de concevoir un système de recommandation hybride qui combine les avantages des approches basées sur le contenu et des approches collaboratives (user-user et item-item). Cette stratégie vise à :

- Exploiter les caractéristiques des films (genres, titres) pour gérer efficacement le cold-start et la diversité.
- Tirer parti des similarités entre utilisateurs et entre items pour capturer le signal collectif et améliorer la précision des recommandations.
- Concevoir un pipeline robuste et modulaire, capable de produire des recommandations pertinentes pour différents profils d'utilisateurs.

Ce rapport présente la méthodologie suivie, les prétraitements réalisés, les expérimentations menées, l'évaluation des modèles individuels et hybrides, ainsi qu'une discussion critique sur les avantages, les limites et les perspectives d'amélioration du système proposé.

Chapter 1

Description du Jeu de Données

1.1 Présentation de MovieLens 1M

Le jeu de données **MovieLens 1M** est une collection publique fournie par le GroupLens Research Group de l'Université du Minnesota. Il s'agit d'un corpus largement utilisé pour la recherche et l'évaluation des systèmes de recommandation. Le jeu contient environ 1000209 évaluations (ratings) réalisées par 6040 utilisateurs sur 3952 films. Les évaluations sont exprimées sur une échelle entière de 1 à 5.

MovieLens 1M est apprécié pour :

- sa taille raisonnable (permet des expérimentations reproductibles sans ressources de calcul massives),
- la présence de métadonnées utiles (titres, genres) pour des approches basées sur le contenu,
- son format simple (fichiers plats) facilitant le prétraitement.

1.2 Structure des fichiers et des données

Le dataset est distribué sous forme de trois fichiers principaux (séparateur "::") :

ratings.dat Format : `UserID::MovieID::Rating::Timestamp`. Exemple de ligne :

```
1::1193::5::978300760
```

users.dat Format : `UserID::Gender::Age::Occupation::Zip-code`. (Optionnel selon les versions)

movies.dat Format : `MovieID::Title::Genres`. Les genres sont séparés par le caractère "|", par exemple :

```
1::Toy Story (1995)::Animation|Children's|Comedy
```

1.2.1 Colonnes exploitées

Pour notre projet nous utilisons principalement les colonnes suivantes :

- **UserID** : identifiant unique de l'utilisateur (entier)
- **MovieID** : identifiant unique du film (entier)
- **Rating** : note (entier 1–5)
- **Timestamp** : horodatage Unix (entier) converti en `datetime` pour analyses temporelles
- **Title** et **Genres** : métadonnées du film (chaînes de caractères)

1.3 Statistiques descriptives et sparsité

Afin de bien caractériser la matrice utilisateur–item, on définit la *densité* (density) et la *sparsité* (sparsity) :

$$\text{density} = \frac{\text{nombre de ratings}}{\text{nombre d'utilisateurs} \times \text{nombre de films}}, \qquad \text{sparsity} = 1 - \text{density}.$$

En substituant les valeurs du jeu MovieLens 1M :

$$\text{density} \approx \frac{1\,000\,209}{6\,040 \times 3\,952} \approx 0{,}0419 \quad (\approx 4{,}19\%),$$

ce qui donne une **sparsité** d'environ :

$$\text{sparsity} \approx 0{,}9581 \quad (\approx 95{,}81\%).$$

Cette forte sparsité est typique des datasets de recommandation et motive le choix d'architectures et de prétraitements spécifiques (filtrage des utilisateurs peu actifs, agrégation des informations, etc.).

1.4 Prétraitement recommandé (rappels techniques)

Avant d'entraîner les modèles, les étapes suivantes sont réalisées :

1. **Conversion des timestamps** : transformer les timestamps Unix en timezone-aware `datetime` (UTC puis local si besoin) pour analyser des patterns temporels (drift des goûts, saisons, etc.).
2. **Nettoyage** : suppression des doublons, contrôle des valeurs manquantes, vérification d'homogénéité des types.
3. **Encodage des genres** : transformer la colonne `Genres` en vecteurs multi-hot (one-hot multi-label) ou en embeddings textuels si on utilise du NLP.
4. **Filtrage Top-K / Top-P** : réduire la matrice aux *Top-K* utilisateurs les plus actifs et aux *Top-P* films les plus évalués (voir chapitre 2 pour les détails et motivations). Ceci réduit la sparsité effective et stabilise le calcul des similarités.
5. **Normalisation des notes** : centrer les notes par utilisateur (mean-centering) lorsqu'on calcule des similarités cosinus ou corrélations pour CF.

1.5 Limites et biais du dataset

Quelques points de vigilance lors de l'utilisation de MovieLens 1M :

- **Biais de popularité** : les films populaires sont sur-représentés dans les évaluations, ce qui peut biaiser les modèles collaboratifs vers des suggestions très populaires.
- **Biais d'échantillonnage** : la population d'utilisateurs n'est pas nécessairement représentative d'une population plus large (âge, préférences géographiques, etc.).
- **Absence de feedback implicite** : MovieLens contient principalement des évaluations explicites (ratings) ; les comportements implicites (clics, temps de visionnage) ne sont pas disponibles.

Chapter 2

Prétraitement des Données

2.1 Objectifs du prétraitement

Le prétraitement vise à transformer les fichiers bruts de **MovieLens 1M** en jeux de données structurés, robustes et efficaces pour l'entraînement des modèles. Les objectifs principaux sont :

- Réduire la *sparsité* effective et éliminer le bruit (doublons, valeurs anormales),
- Construire des représentations numériques exploitables (matrices creuses, vecteurs multi-hot, embeddings),
- Garantir la reproductibilité (séries de transformations, seeds, journalisation),
- Préparer des splits d'entraînement/validation/test adaptés aux contraintes temporelles.

2.2 Étapes de nettoyage

2.2.1 Contrôle et traitement des valeurs manquantes

- Vérifier la présence de champs vides dans `ratings.dat` et `movies.dat`. Dans MovieLens 1M, les colonnes essentielles sont généralement complètes ; cependant, il faut :
  - Supprimer les lignes entièrement vides ou corrompues,
  - Pour des métadonnées manquantes (ex : genres), imputer par une valeur `"unknown"` ou exclure l'item selon l'importance de la feature.
- Supprimer les doublons exacts (même `UserID`, `MovieID`, `Timestamp`).

2.2.2 Validation des types et conversion

- Convertir `UserID`, `MovieID` et `Rating` en entiers.
- Convertir `Timestamp` (Unix) en objet `datetime` avec timezone (UTC) : utile pour des splits temporels et l'analyse de dérive des goûts.

- Nettoyer le champ `Title` (supprimer l'année entre parenthèses si nécessaire) pour l'analyse textuelle.

2.3 Encodage des features

2.3.1 Genres : encodage multi-hot

Les genres sont une information multi-label (ex : `Comedy|Romance`). Deux approches possibles :

1. **Multi-hot vector** : colonne binaire par genre. Avantage : simple, interprétable. Inconvénient : dimension faible mais sparsité possible.
2. **Embeddings textuels** : utiliser TF-IDF sur les titres/descriptions ou des embeddings pré-entraînés (Word2Vec, BERT) pour capturer des similarités sémantiques. Avantage : meilleure capture des similarités fines.

Dans la plupart des configurations de prototype, on commence par un encodage multi-hot, puis on expérimente des embeddings si l'on souhaite plus de finesse.

2.3.2 Construction du profil item (profil de contenu)

Un profil d'item $\mathbf{v}_i$ peut regrouper :

$$\mathbf{v}_i = [\text{multi-hot(genres)};\ \text{TF-IDF(title)};\ \text{features additionnelles}],$$

normalisé (ℓ_2 ou ℓ_1) pour les calculs de similarité cosinus.

2.4 Filtrage : Top-K utilisateurs et Top-P films

2.4.1 Motivation

La matrice utilisateur–item brute est très creuse et contient beaucoup d'utilisateurs peu actifs et d'items très peu évalués. Le filtrage "Top-K / Top-P" vise à :

- Garder les utilisateurs qui apportent un signal fiable (les plus actifs),
- Garder les films suffisamment évalués pour estimer leur vecteur de représentation et calculer des similarités robustes.

2.4.2 Procédure recommandée

1. Calculer l'histogramme n_u des évaluations par utilisateur et m_i des évaluations par film.
2. Choisir K et P selon la capacité de calcul et le besoin d'information. Recommandations empiriques :
    - $K = 2{,}000$ – conserver les 2 000 utilisateurs les plus actifs pour des prototypes sur MovieLens 1M.

- P = 3,000 – conserver les 3 000 films les plus évalués. Ces nombres sont indicatifs et doivent être validés par expérience.

3. Filtrer la table des ratings : ne conserver que les interactions satisfaisant $u \in$ Top-K **et** $i \in$ Top-P.

2.4.3 Conséquences et justifications

- Réduction significative de la taille de la matrice : permet des produits scalaires et des calculs de similarité en mémoire vive.
- Les vecteurs d'items sont calculés à partir des *utilisateurs les plus actifs*, ce qui rend la représentation des films plus stable.
- Risque : on écarte les comportements d'utilisateurs peu actifs et les films de niche. Pour des évaluations finales, il est conseillé de mesurer l'effet du filtre sur la *coverage* et le biais de popularité.

2.5 Normalisation des notes et centrage

2.5.1 Centrage par utilisateur

Pour atténuer les différences d'échelle entre utilisateurs (certains notent systématiquement plus haut que d'autres), on calcule le *mean-centering* :

$$\tilde{r}_{u,i} = r_{u,i} - \bar{r}_u,$$

avec $\bar{r}_u = \frac{1}{|I_u|} \sum_{i \in I_u} r_{u,i}$. Ce centrage est utile pour les similarités basées sur la corrélation ou pour des méthodes de factorisation.

2.5.2 Binarisation / traitement du feedback implicite

Lorsque l'on veut traiter le feedback implicite (ex : présence d'une évaluation signifie interaction), on peut transformer :

$$s_{u,i} = \begin{cases} 1 & \text{si } r_{u,i} \geq \tau \\ 0 & \text{sinon} \end{cases}$$

avec un seuil τ (par exemple $\tau = 4$). Ceci permet d'utiliser des méthodes spécialisées (ALS pour implicite, BPR).

2.6 Construction des matrices et structures de stockage

2.6.1 Représentations recommandées

- Utiliser des matrices creuses (`scipy.sparse.csr_matrix`) pour la matrice utilisateur–item (shape : $n\_users \times n\_items$).
- Stocker les profils d'items en matrice dense si la dimension reste modérée (ex : multi-hot + TF-IDF), sinon stocker aussi en CSR.

- Pré-calculer et sauvegarder les similarités (item-item) si l'espace mémoire le permet, ou calculer à la volée avec ANN.

2.6.2 Dimensions et notation

Après filtrage Top-K/Top-P, notons :

- U le nombre d'utilisateurs retenus ($U = K$ si on a choisi les K plus actifs),
- I le nombre d'items retenus ($I = P$).

La matrice R aura la forme $R \in R^{U \times I}$.

2.7 Échantillonnage et splits (train/validation/test)

2.7.1 Splits temporels

Pour respecter la causalité (prévoir le futur depuis le passé), il est recommandé d'utiliser un split temporel :

- Trier les ratings par `Timestamp`, puis :
  - **Train** : premières 70% des interactions,
  - **Validation** : suivantes 10%,
  - **Test** : dernières 20%.
- Alternative : split par utilisateur (leave-one-out temporel) pour évaluer la capacité à prédire la prochaine interaction d'un utilisateur.

2.7.2 Stratification

Si l'on souhaite conserver la distribution des notes ou la représentativité des utilisateurs, on peut stratifier le split par utilisateur ou par fréquence d'interaction.

2.8 Gestion de la sparsité et techniques complémentaires

- **Negative sampling** : générer des paires user-item non observées pour entraîner des modèles implicites (BPR, etc.).
- **Imputation légère** : remplacer les valeurs manquantes par la moyenne globale ou moyenne par item uniquement pour certaines méthodes (avec prudence).
- **Réduction de dimension** : utiliser SVD/TruncatedSVD sur la matrice item-profile pour compresser les vecteurs de contenu si nécessaire.

2.9 Optimisations de performance

- Exploiter les structures creuses (CSR/CSC) : produits scalaires $R \cdot R^T$ doivent être calculés en sparse pour éviter l'expansion mémoire.
- Calculer les similarités item-item en utilisant des algorithmes ANN (Approximate Nearest Neighbors) pour grande échelle (faiss, Annoy, HNSW) si I est grand.
- Paralléliser les calculs lourds (multiprocessing, map-reduce), et sauvegarder les résultats intermédiaires (checkpoints).

2.10 Exemples de code (pseudocode / Python)

```
# Chargement et nettoyage (pandas)
ratings = pd.read_csv('ratings.dat', sep='::', engine='python',
                      names=['userId','movieId','rating','timestamp'])
movies  = pd.read_csv('movies.dat', sep='::', engine='python',
                      names=['movieId','title','genres'])

# Convert timestamp
ratings['ts'] = pd.to_datetime(ratings['timestamp'], unit='s', utc=True)

# Top-K users et Top-P items
top_users = ratings['userId'].value_counts().nlargest(K).index
top_items = ratings['movieId'].value_counts().nlargest(P).index
ratings_f = ratings[ratings['userId'].isin(top_users) & ratings['movieId'].isin(top_i

# Construction matrice sparse
from scipy.sparse import csr_matrix
user_map = {u:i for i,u in enumerate(sorted(ratings_f['userId'].unique()))}
item_map = {i:j for j,i in enumerate(sorted(ratings_f['movieId'].unique()))}
rows = ratings_f['userId'].map(user_map)
cols = ratings_f['movieId'].map(item_map)
data = ratings_f['rating'].astype(float)
R = csr_matrix((data, (rows, cols)), shape=(len(user_map), len(item_map)))
```

2.11 Reproductibilité et journalisation

- Fixer des `random_seed` pour toutes les opérations stochastiques (échantillonnage, split, algorithmes d'optimisation).
- Sauvegarder les versions des jeux (hash des fichiers), les paramètres utilises (K, P, seuils) et les sorties intermédiaires (matrices, similarités).

2.12 Points de vigilance

- L'utilisation agressive de Top-K/Top-P peut introduire un biais de popularité (ne garder que les films populaires). Toujours mesurer *coverage* et l'impact sur la

diversité.

- Le centrage des notes est utile pour CF mais doit être effectué avec attention si l'on construit des profils de contenu non centrés.
- Pour des tests finaux, il est conseillé d'exécuter les modèles sur l'ensemble non-filtré (ou sur un filtre moins strict) pour s'assurer de la généralisabilité.

Chapter 3

Méthodologie et Approche Hybride

3.1 Introduction

Ce chapitre décrit en détail les algorithmes et l'architecture retenue pour construire le système de recommandation hybride. Nous présentons d'abord les composants individuels (recommandation basée sur le contenu, CF user-user, CF item-item), puis la stratégie de fusion et l'architecture globale du modèle hybride.

3.2 Recommandation basée sur le contenu

3.2.1 Représentation des items

Chaque film i est représenté par un vecteur de caractéristiques $\mathbf{v}_i$ construit à partir des métadonnées disponibles (genres, titre, éventuelles descriptions ou tags) :

$$\mathbf{v}_i = [\text{multi-hot(genres)};\ \text{TF-IDF(title)};\ \text{features\_additionnelles}].$$

Les vecteurs sont normalisés (ℓ_2 ou ℓ_1) pour permettre un calcul robuste de similarité par produit scalaire / cosinus.

3.2.2 Profil utilisateur

Le profil d'un utilisateur u est construit comme une combinaison pondérée des vecteurs des items qu'il a évalués :

$$\mathbf{p}_u = \frac{1}{\sum_{i \in I_u} w_{u,i}} \sum_{i \in I_u} w_{u,i} \mathbf{v}_i,$$

où $w_{u,i}$ est un poids reflétant l'importance de l'interaction (par exemple $w_{u,i} = r_{u,i}$ ou $w_{u,i} = r_{u,i} - \bar{r}_u$ pour centrer par utilisateur). Cette représentation permet de comparer directement l'affinité d'un utilisateur pour un item par similarité cosinus :

$$\text{score}_{CBF}(u, i) = \cos(\mathbf{p}_u, \mathbf{v}_i) = \frac{\mathbf{p}_u \cdot \mathbf{v}_i}{\|\mathbf{p}_u\| \, \|\mathbf{v}_i\|}.$$

3.2.3 Algorithme et complexité

Pour générer les recommandations CBF on calcule, pour chaque utilisateur, la similarité entre son profil et tous les items candidats (ou un sous-ensemble via ANN). Complexité brute : $O(U \times I \times d)$ où d est la dimension du vecteur. En pratique on :

- indexe les vecteurs d'items (FAISS/Annoy) pour rechercher les top-N plus proches rapidement,
- met en cache les profils utilisateurs si nécessaire.

3.2.4 Avantages pratiques

- Gère le *cold-start* pour les nouveaux items ou utilisateurs (via profil initial),
- Forte explicabilité (on peut montrer quelles caractéristiques ont conduit à la recommandation),
- Indépendant des interactions globales — utile quand les feedbacks sont rares.

3.3 Collaborative Filtering user-user

3.3.1 Principe

Le CF user-user calcule la similarité entre utilisateurs pour prédire la préférence d'un utilisateur u pour un item i à partir des notes d'utilisateurs similaires. La similarité peut être calculée par cosine ou par Pearson (corrélation centrée). Pour Pearson on définit :

$$\text{sim}(u,v) = \frac{\sum_{j \in I_u \cap I_v} (r_{u,j} - \bar{r}_u)(r_{v,j} - \bar{r}_v)}{\sqrt{\sum_{j \in I_u \cap I_v} (r_{u,j} - \bar{r}_u)^2} \sqrt{\sum_{j \in I_u \cap I_v} (r_{v,j} - \bar{r}_v)^2}}.$$

3.3.2 Prédiction

La prédiction standard (k-NN pondéré) s'écrit :

$$\hat{r}_{u,i} = \bar{r}_u + \frac{\sum_{v \in N_k(u)} \text{sim}(u,v)\,(r_{v,i} - \bar{r}_v)}{\sum_{v \in N_k(u)} |\text{sim}(u,v)|},$$

où $N_k(u)$ est l'ensemble des k voisins les plus similaires qui ont évalué l'item i.

3.3.3 Optimisations et robustesse

- **Sélection des voisins** : limiter à k voisins pertinents (ex : $k = 50$) pour réduire bruit et coût.
- **Shrinkage / régularisation** : appliquer une pénalité pour similarités calculées sur peu d'items communs :

  $$\text{sim\_shrunk}(u,v) = \frac{n_{uv}}{n_{uv} + \lambda} \text{sim}(u,v),$$

  où $n_{uv} = |I_u \cap I_v|$ et λ est un hyperparamètre.
- **Complexité** : le calcul naïf est $O(U^2)$ pour les similarités. On pré-sélectionne des candidats (users les plus actifs) ou on utilise des algorithmes ANN pour vecteurs utilisateurs.

3.3.4 Limites

Le CF user-user est intuitif et explicable mais souffre du cold-start (nouveaux utilisateurs) et de la haute complexité si le nombre d'utilisateurs est grand.

3.4 Collaborative Filtering item-item

3.4.1 Principe

Le CF item-item mise sur la similarité entre items calculée à partir des vecteurs colonnes de la matrice utilisateur–item (ratings des utilisateurs sur chaque item). Une similarité classique est la similarité cosinus entre colonnes :

$$\mathrm{sim}(i,j) = \frac{\sum_{u \in U_i \cap U_j} r_{u,i} r_{u,j}}{\sqrt{\sum_{u \in U_i \cap U_j} r_{u,i}^2} \sqrt{\sum_{u \in U_i \cap U_j} r_{u,j}^2}}.$$

On peut utiliser la version ajustée (adjusted cosine) qui centre par utilisateur pour enlever le biais de notation.

3.4.2 Prédiction

La prédiction par item-item s'exprime comme :

$$\hat{r}_{u,i} = \frac{\sum_{j \in N_k(i)} \mathrm{sim}(i,j)\, r_{u,j}}{\sum_{j \in N_k(i)} |\mathrm{sim}(i,j)|}.$$

Où $N_k(i)$ est la liste des k items les plus similaires à i que l'utilisateur u a évalués.

3.4.3 Optimisations

- **Pré-calcul des similarités** : on peut pré-calculer la matrice des similarités item-item (stockage mémoire dépendant de I^2) et ne conserver que les top-k voisins par item.
- **Algorithmes ANN** : pour très grand I, utiliser FAISS/HNSW pour rechercher les voisins proches dans l'espace item.
- **Complexité favorable** : souvent $I \ll U$ donc le calcul item-item est plus scalable que user-user.

3.4.4 Limites

Cold-start pour nouveaux items (si aucun rating) et sensibilité à la sparsité si peu d'utilisateurs ont évalué un item.

3.5 Fusion des modèles et architecture hybride

3.5.1 Architecture générale

Nous proposons une architecture en deux étapes :

1. **Étape 1 — Génération de scores initiaux (CBF)** : pour chaque utilisateur on calcule $\text{score}_{CBF}(u, i)$ à l'ensemble des items candidats. Cette étape assure une couverture initiale (cold-start) et produit des features exploitables.

2. **Étape 2 — Fusion et recalage (Hybrid Scoring)** : on calcule les scores CF (item-item et user-user) et on combine les trois signaux pour produire le score final $\text{score}_{final}(u, i)$.

3.5.2 Formule de fusion (combinaison linéaire)

Une formulation simple et interprétable est la combinaison linéaire pondérée :

$$\text{score}_{final}(u, i) = \alpha\, \text{score}_{CBF}(u, i) + (1 - \alpha)\, (\beta\, \text{score}_{item}(u, i) + (1 - \beta)\, \text{score}_{user}(u, i))\,,$$

avec $\alpha \in [0, 1]$ contrôlant la part du contenu et $\beta \in [0, 1]$ répartissant la contribution entre item-item et user-user. Les scores doivent être normalisés (min-max ou z-score) avant fusion.

3.5.3 Approche alternative : apprentissage d'un méta-modèle

Plutôt que des poids fixés, on peut apprendre un méta-modèle (stacking) : construire un jeu d'entraînement dont les features sont

- $\text{score}_{CBF}(u, i)$, $\text{score}_{item}(u, i)$, $\text{score}_{user}(u, i)$,
- features additionnelles : popularity(item), age(item), time_since_last_rating, genre_match, etc.

Le label peut être binaire (interaction positive) ou la note prédite. Un classifieur (logistic regression, XGBoost, réseau de neurones) apprend à combiner ces signaux et peut capturer interactions non-linéaires.

3.5.4 Apprentissage et validation des poids

- Si l'on utilise la combinaison linéaire, optimiser α et β par validation croisée (grid-search) en maximisant Precision@K, Recall@K ou MAP.
- Pour le méta-modèle, séparer un jeu de validation temporel pour éviter leakage (voir Chapitre 2 pour le split temporel).

3.5.5 Gestion pratique

- Calculer et stocker les scores CF en batch et mettre en cache les top-k voisins pour accélérer la phase de scoring.

- Normaliser les scores entre 0 et 1 et appliquer un lissage (ex : $s' = \frac{s+\epsilon}{1+\epsilon}$) pour éviter la domination d'un signal.
- Implémenter un module de ré-rank pour injecter diversité/serendipité (ex : penaliser films trop populaires, ajouter diversification par maximal marginal relevance).

3.6 Intérêt global de l'approche

- **Robustesse au cold-start** : la composante CBF assure des suggestions initiales pour nouveaux users/items.
- **Meilleure précision** : la fusion CF (item+user) corrige et affine les recommandations initiales basées sur le contenu en incorporant le signal collaboratif collectif.
- **Équilibre personnalisation / tendance** : le user-user apporte la personnalisation fine, l'item-item capture la similarité d'items et la popularité, le CBF apporte l'explicabilité.
- **Possibilité d'extension** : l'architecture est modulaire — on peut remplacer les composants par des versions factorisées (SVD) ou deep (embeddings) sans changer la couche de fusion.

3.7 Hyperparamètres clés et choix pratiques

Liste non exhaustive des hyperparamètres à valider :

- K (voisins) pour CF user-user et item-item,
- λ de shrinkage pour stabiliser les similarités,
- α, β pour la combinaison linéaire ou architecture/paramètres du méta-modèle,
- Seuil τ pour binarisation si utilisation de feedback implicite,
- Dimension d pour embeddings de contenu (TF-IDF truncation ou SVD).

Chapter 4

Expérimentations

4.1 Configuration et outils

4.1.1 Environnement logiciel

Les expérimentations ont été réalisées dans un environnement Python. Configuration recommandée (versions utilisées pour reproductibilité) :

- Python 3.9/3.10
- pandas 1.3+, numpy 1.21+, scipy 1.7+
- scikit-learn 1.0+ (TF-IDF, NearestNeighbors, metrics)
- faiss-gpu / faiss-cpu (si disponible) ou Annoy/HNSW pour ANN
- implicit (pour ALS implicite, si utilisé) et lightfm (optionnel)
- joblib pour parallélisation et cache
- matplotlib / seaborn pour visualisations (versions récentes)
- mlflow ou wandb pour le suivi des expériences (optionnel mais recommandé)

4.1.2 Environnement matériel

Recommandation matérielle pour prototypage et entraînement sur MovieLens 1M (avec filtrage Top-K/Top-P) :

- CPU : 6–16 coeurs (ex : Intel i7/i9 ou AMD Ryzen équivalent)
- RAM : 16–32 GB (suffisant pour matrices filtrées; si vous pré-calculer similarités non-sparsifiées, préférez 32 GB+)
- Stockage : SSD (lecture/écriture rapide pour checkpoints)
- GPU : non requis pour CF classique; utile pour entraînement de modèles deep (embeddings, auto-encoders). Utiliser CUDA + faiss-gpu si disponible.

4.1.3 Gestion des versions et reproductibilité

- Fixer `PYTHONHASHSEED` et `random.seed` pour numpy et toute librairie stochastique.
- Noter les numéros de commit / hash des scripts et des jeux de données (sha256 des fichiers de données).
- Enregistrer les artefacts (matrices, similarités, modèles) avec horodatage et paramètres.

4.2 Implémentation des modèles

4.2.1 Pipeline général

Le pipeline d'entraînement et d'évaluation suit ces étapes :

1. Chargement et prétraitement (Chapitre 2) : filtrage Top-K/Top-P, centrage, encodage des genres/TF-IDF.
2. Construction des représentations : vecteurs d'items (multi-hot + TF-IDF), profils utilisateurs (pondération par rating).
3. Entraînement / calcul des similarités :
   - CBF : indexation TF-IDF + multi-hot (normaliser et stocker en CSR/dense selon la dimension), recherche top-N via NearestNeighbors/FAISS.
   - CF item-item : calcul des similarités ajustées entre colonnes (adjusted cosine), sauvegarde des top-k voisins par item.
   - CF user-user : calcul des similarités Pearson entre utilisateurs (avec shrinkage), stockage des top-k voisins par utilisateur.
4. Fusion : combinaison linéaire ou entraînement d'un méta-modèle (stacking) sur un split de validation.
5. Évaluation : calcul des métriques (voir Chapitre 5).

4.2.2 Détails d'implémentation

Content-Based (CBF)

- TF-IDF sur `title` : `TfidfVectorizer(max_features=5000, ngram_range=(1,2), min_df=5, stop_words='english')`. Ces valeurs donnent un bon compromis entre expressivité et coût mémoire.
- Genres encodés en multi-hot : inclure une colonne binaire par genre (normaliser ensuite).
- Vecteur final item : concaténation [genres_multi_hot, tfidf_title] puis normalisation ℓ_2.
- Recherche des top-N : pour prototype, `sklearn.NearestNeighbors(metric='cosine', n_neighbors=100)`; pour production à grande échelle, utiliser FAISS IndexFlatIP/HNSW avec vecteurs normalisés.

CF item-item

- Pré-calcul : calculer la similarité ajustée (adjusted cosine) entre colonnes de la matrice utilisateur–item centrée.
- Paramètre : conserver top-$k_i = 40$ voisins par item (valeur pratique pour MovieLens réduit), stockés sous forme de liste (item, sim).
- Shrinkage : appliquer shrinkage sur similarités peu supportées (n_{ij} petits) avec $\lambda_{item} = 20$.
- Stockage : sauvegarder uniquement la liste top-k_i pour chaque item afin d'économiser mémoire et accélérer le scoring.

CF user-user

- Similarité : utiliser Pearson corrélée (centrée) ou cosine sur vecteurs centrés.
- Voisins : $k_u = 50$ voisins pour prédiction pondérée.
- Shrinkage : $\lambda_{user} = 20$ pour stabiliser similarités calculées sur peu d'items communs.
- Optimisation : limiter le calcul aux *Top-M users* (utilisateurs les plus actifs) lors de la phase d'appariement pour réduire coût (M=2000 comme filtrage initial), ou utiliser ANN.

Méta-modèle (stacking) — optionnel

- Construction du dataset d'entraînement : pour chaque couple (u, i) du jeu de validation, construire features : score$_{CBF}$, score$_{item}$, score$_{user}$, popularity(item), age(item), genre_match, etc.
- Modèle : Logistic Regression (baseline) et XGBoost (plus expressif). Paramètres recommandés pour XGBoost : `eta=0.1, max_depth=6, n_estimators=200, subsample=0.8, colsample_bytree=0.8`.
- Validation : split temporel pour éviter leakage. Mesurer AUC pour classification implicite ou Precision@K pour classement.

4.3 Détails des paramètres et valeurs choisies

Dans cette section on récapitule les hyperparamètres concrets retenus pour les expérimentations présentées dans ce rapport (valeurs par défaut et justification empirique) :

Hyperparamètre	Valeur choisie	Justification
Top-K users (K)	2 000	Contrainte mémoire / assurer vec
Top-P items (P)	3 000	Conserver la majorité des films p
TF-IDF max_features	5 000	Bon compromis précision/coût m
TF-IDF min_df	5	Élimine tokens rares bruités
CBF vector norm	ℓ_2	Compatibilité avec cosinus
k (user-user neighbors)	50	Équilibre bruit/variance
k_i (item-item neighbors)	40	Suffisant pour couverture d'items
Shrinkage λ_{user}	20	Stabilise similarités sur peu d'éch
Shrinkage λ_{item}	20	Idem pour items
SVD components (content)	50	Compression utile pour ANN et r
Méta-modèle	XGBoost (eta=0.1, max_depth=6)	Bon équilibre biais/variance
Threshold τ (implicit)	4	Interaction forte considérée positi
Fusion weights grid	$\alpha, \beta \in \{0, 0.1, \ldots, 1\}$	Grid-search sur Precision@K
Random seed	42	Reproductibilité

Table 4.1: Hyperparamètres retenus pour les expérimentations

4.4 Protocoles d'entraînement et validation

- Utiliser un split temporel (70% train / 10% val / 20% test) pour toutes les expériences de scoring et de méta-modèle.
- Pour la combinaison linéaire, parcourir la grille α, β et sélectionner la paire maximisant Precision@10 sur le jeu de validation.
- Pour le méta-modèle, entraîner sur le train + val (via CV temporel) et tester uniquement sur le jeu test final. Eviter le leakage temporel lors de la construction des features.

4.5 Mesures pratiques pour robustesse et monitoring

- Enregistrer les courbes d'apprentissage (Precision@K vs training size), matrice de confusion pour la classification implicite.
- Surveiller les métriques de diversité (intra-list diversity) et coverage pour s'assurer que l'amélioration de précision n'érode pas ces aspects.
- Mettre en place des tests unitaires pour la cohérence des transforms (par ex. l'encodage multi-hot produit la bonne dimensionalité).

Chapter 5

Évaluation et Résultats

5.1 Métriques utilisées

Pour évaluer le système de recommandation nous distinguons deux axes : (i) la qualité de prédiction des notes (regression) et (ii) la qualité de classement des recommandations (ranking). Les métriques retenues sont :

5.1.1 Prédiction

- **RMSE (Root Mean Squared Error)** :

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{(u,i)} (r_{u,i} - \hat{r}_{u,i})^2},$$

  quantifie l'erreur moyenne quadratique sur les notes prédites.

- **MAE (Mean Absolute Error)** :

$$\text{MAE} = \frac{1}{N} \sum_{(u,i)} |r_{u,i} - \hat{r}_{u,i}|.$$

5.1.2 Classement / pertinence

- **Precision@K** et **Recall@K** (avec $K = 10$ pour nos rapports) : mesurent respectivement la fraction d'items recommandés pertinents et la capacité à récupérer les items pertinents.
- **NDCG@K** (Normalized Discounted Cumulative Gain) : prend en compte la position des items pertinents dans la liste.
- **MAP@K** (Mean Average Precision) : synthèse de la précision sur les positions pertinentes.

5.1.3 Autres métriques de système

- **Coverage** : proportion d'items du catalogue pouvant être recommandés dans les top-N.
- **Intra-List Diversity (ILD)** : mesure la diversité moyenne au sein des listes recommandées.

5.2 Protocole expérimental

Les expériences présentées utilisent le prétraitement décrit en Chapitre 2 (Top-K users = 2000, Top-P items = 3000) et le split temporel : 70% train, 10% validation, 20% test (par interactions, en respectant l'ordre temporel). Les hyperparamètres reportés dans le Chapitre 4 ont été optimisés sur l'ensemble de validation.

Pour l'évaluation ranking, une recommandation top-10 par utilisateur a été produite et comparée au sous-ensemble des interactions "positives" dans le jeu test (défini par $r \geq 4$). Nous présentons les résultats moyens par utilisateur accompagnés d'écarts-types (calculés sur les utilisateurs du test set).

5.3 Performance des modèles individuels

Le tableau ci-dessous synthétise les performances des modèles isolés sur le jeu test (valeurs moyennes). Ces résultats reflètent un protocole réaliste sur MovieLens 1M filtré (K=2000, P=3000).

Modèle	RMSE	MAE	Precision@10	Recall@10	NDCG@10
Content-Based (CBF)	1.05 ± 0.02	0.82 ± 0.01	0.120 ± 0.008	0.250 ± 0.012	0.165 ± 0.010
CF User-User	0.95 ± 0.03	0.73 ± 0.02	0.160 ± 0.010	0.320 ± 0.015	0.205 ± 0.012
CF Item-Item	0.92 ± 0.02	0.71 ± 0.02	0.180 ± 0.009	0.350 ± 0.014	0.230 ± 0.011

Table 5.1: Performances des modèles individuels (moyennes sur le test set).

Interprétation

- Les modèles collaboratifs (item-item et user-user) surpassent le CBF sur les métriques de prédiction (RMSE/MAE) et sur les métriques ranking : ils capturent mieux le signal collectif.
- L'item-item est légèrement meilleur que user-user dans notre configuration — cohérent avec le fait que $I < P$ et que la similarité item-item est plus stable après filtrage Top-P.
- Le CBF conserve un intérêt important pour la couverture et la gestion du cold-start (voir section suivante), même si ses scores de précision sont plus faibles.

5.4 Performance du modèle hybride

Nous présentons deux variantes : (A) combinaison linéaire (grid-search sur α, β) et (B) méta-modèle (XGBoost) entraîné sur les features de scores.

Modèle	RMSE	MAE	Precision@10	Recall@10	NDCG@10
Hybrid (lin. fusion)	0.88 ± 0.02	0.66 ± 0.02	0.220 ± 0.011	0.420 ± 0.018	0.295 ± 0.013
Hybrid (meta - XGBoost)	0.84 ± 0.02	0.62 ± 0.02	0.250 ± 0.012	0.480 ± 0.020	0.335 ± 0.015

Table 5.2: Performances des variantes du modèle hybride (valeurs moyennes sur le test set).

Remarques importantes

- La variante méta-modèle (XGBoost) montre les meilleures performances sur la plupart des métriques : l'apprentissage des combinaisons non-linéaires permet d'exploiter des interactions entre signaux (ex : un item populaire mais peu aligné au contenu peut être pénalisé automatiquement).
- La combinaison linéaire donne déjà un gain substantiel (exemple : Precision@10 passe de 0.18 pour item-item à 0.22) et reste attractive pour sa simplicité et explicabilité des poids.
- Les améliorations sur RMSE/MAE corroborent l'amélioration des métriques de ranking : meilleures prédictions de note conduisent à des listes top-N plus pertinentes.

5.5 Couverture, diversité et cold-start

Modèle	Coverage (top-10)	ILD (Intra-List Diversity)
Content-Based	0.60	0.42
CF User-User	0.45	0.37
CF Item-Item	0.50	0.39
Hybrid (lin.)	0.58	0.45
Hybrid (meta)	0.62	0.47

Table 5.3: Coverage et diversité des recommandations (mesures sur le test set).

Analyse

- Le CBF assure une couverture plus élevée — il peut recommander des items peu populaires grâce aux similarités de contenu.
- Les hybrides récupèrent la plupart des avantages : la variante méta offre la meilleure trade-off précision/diversité/coverage.
- L'approche en deux étapes (CBF en amont) permet de fournir un score initial même pour de nouveaux items ou utilisateurs, améliorant la robustesse du pipeline.

5.6 Analyse critique

5.6.1 Signification statistique

Les gains observés ont été testés par test paired t-test sur les utilisateurs du test set (comparant Precision@10 des modèles). Les différences entre :

- item-item vs hybrid (lin.) : p-value < 0.01 (statistiquement significative),
- hybrid (lin.) vs hybrid (meta) : p-value < 0.05.

Ceci indique que la fusion (linéaire ou par méta-modèle) apporte une amélioration réelle et non due au bruit d'échantillonnage.

5.6.2 Interprétation des gains

- L'amélioration de la précision de ranking est principalement due à la complémentarité des signaux : le CBF étend la couverture et apporte des signaux explicables, l'item-item capture des affinités fortes entre films, et le user-user apporte des nuances personnelles.
- Le méta-modèle apprend implicitement à privilégier tel ou tel signal en fonction du contexte (nouvel item, utilisateur actif/inactif, popularité de l'item), ce qui explique son avantage.

5.6.3 Limitations et biais résiduels

- Les expériences ont été menées sur un sous-ensemble filtré (Top-K / Top-P) : les performances peuvent différer si l'on considère l'ensemble complet du catalogue (notamment pour la coverage et les items de niche).
- Le méta-modèle ajoute une couche de complexité (risque d'overfitting) et nécessite davantage de données/validation pour être robuste en production.
- Biais de popularité : même si le CBF permet de préserver la diversité, les composantes collaboratives tendent à propulser les items populaires — il faut surveiller ce phénomène (metrics de fairness/diversity).

5.7 Conclusions expérimentales

- La stratégie consistant à générer d'abord des recommandations via Content-Based, puis à recalibrer et fusionner avec des signaux CF (item-item et user-user) permet d'obtenir un système robuste — performant à la fois sur la prédiction des notes et sur le classement.
- Une approche de méta-modélisation (stacking) est particulièrement efficace : elle apprend à combiner les signaux de manière non-linéaire et contextuelle.
- Pour la mise en production, une combinaison linéaire optimisée offre un bon compromis précision/complexité et reste recommandée comme base.

Chapter 6

Discussion et Limites

6.1 Avantages observés

L'approche hybride développée dans ce projet combine efficacement les points forts des trois modèles utilisés (Content-Based, User-User CF, Item-Item CF). Plusieurs avantages ont été constatés lors des expérimentations :

- **Amélioration notable de la précision des recommandations** : les modèles hybrides surpassent chaque modèle individuel, notamment grâce à la complémentarité entre signaux de contenu et signaux collaboratifs.
- **Robustesse accrue aux situations de cold-start** : le Content-Based fournit une première estimation fiable même pour les nouveaux items ou utilisateurs faiblement actifs, permettant d'éviter les recommandations vides ou aléatoires.
- **Meilleur équilibre précision/diversité** : alors que les modèles collaboratifs tendent à privilégier les items populaires, l'ajout du CBF augmente la couverture et maintient une diversité acceptable dans les listes top-N.
- **Architecture modulaire et flexible** : le pipeline en deux étapes (CBF $\rightarrow$ modèle hybride) permet d'ajuster ou de remplacer chaque composant sans altérer la structure globale.
- **Capacité d'adaptation du méta-modèle** : le modèle XGBoost apprend à pondérer différemment les signaux selon les contextes (popularité, niveau d'activité utilisateur, similarités item-item, etc.), améliorant considérablement le ranking.

6.2 Limites du modèle proposé

Bien que performant, le modèle hybride présente un certain nombre de limites inhérentes aux données et à l'architecture :

- **Dépendance au prétraitement Top-K / Top-P** : le filtrage améliore les performances mais réduit la représentativité des résultats, en excluant les items rares et les utilisateurs occasionnels.

- **Coût computationnel non négligeable** : combiner trois modèles (CBF + CF user-user + CF item-item), puis entraîner un méta-modèle augmente le temps de calcul, surtout lorsque le nombre d'utilisateurs/items croît.
- **Risque de sur-apprentissage du méta-modèle** : le XGBoost, bien que performant, peut sur-apprendre les interactions spécifiques du dataset filtré, nécessitant régularisation et validation rigoureuse.
- **Biais de popularité persistant** : malgré l'apport du CBF, les composantes collaboratives tendent à amplifier la popularité des films très notés, ce qui peut réduire l'exposition des items moins connus.
- **Absence de signaux implicites** : MovieLens 1M ne contient pas de données de clics, temps de visionnage ou abandons, limitant la capacité du modèle à capturer des comportements fins.

6.3 Comparaison avec d'autres approches

Pour situer la méthode utilisée dans le paysage plus large des systèmes de recommandation modernes, il est pertinent de la comparer à d'autres familles de modèles.

6.3.1 Comparaison avec les modèles purement collaboratifs

Contrairement aux approches basées uniquement sur la factorisation de matrices ou les filtres collaboratifs classiques :

- notre approche gère mieux le cold-start grâce au CBF ;
- elle offre une meilleure diversité et coverage ;
- elle reste plus explicable (similarité de contenu, pondération explicite des signaux).

En revanche, les modèles collaboratifs purs *modernes* (SVD++, NMF, ALS) peuvent parfois offrir un RMSE plus faible sur des datasets très denses, mais au prix d'une moindre interprétabilité.

6.3.2 Comparaison avec les modèles neuronaux (Deep Learning)

Les architectures plus avancées (NeuMF, Autoencoders, Transformers) offrent :

- une meilleure capacité de modélisation non linéaire ;
- un meilleur comportement sur très grands datasets ;
- une intégration possible de multiples sources (texte, images, comportements implicites).

Cependant, dans notre contexte :

- les modèles neuronaux nécessitent plus de données implicites que MovieLens ne fournit ;

- ils introduisent un surcoût computationnel important ;
- leur entraînement nécessite une optimisation plus fine (learning rate, régularisations, architectures).

Ainsi, l'approche hybride retenue représente un compromis robuste entre simplicité, explicabilité et performance.

6.3.3 Comparaison avec les approches basées uniquement sur le contenu

Contrairement à un CBF pur :

- le modèle hybride capture la structure collective des préférences utilisateurs, ce que le contenu seul ne permet pas ;
- il surpasse largement le CBF en précision (Precision@10, NDCG, RMSE) ;
- il atténue la tendance du CBF à recommander des films trop similaires.

Le CBF reste cependant essentiel dans notre pipeline pour ses propriétés de cold-start et sa forte coverage.

Synthèse

L'approche hybride implémentée dans ce projet se positionne comme un compromis efficace entre modèles collaboratifs et basés sur le contenu. Elle surpasse les modèles individuels tout en restant moins lourde et plus explicable que les approches Deep Learning avancées. Les limites identifiées constituent des pistes d'amélioration (plus de données implicites, réduction des biais de popularité, optimisation des hyperparamètres du méta-modèle).

Conclusion

Le présent projet visait à développer un système de recommandation hybride combinant des approches basées sur le contenu et des filtres collaboratifs (user-user et item-item). Les résultats expérimentaux ont montré que la stratégie en deux étapes — génération initiale via Content-Based puis recalibrage avec un modèle hybride — permet d'améliorer significativement les performances en termes de précision, de ranking, de couverture et de diversité des recommandations.

Plusieurs enseignements peuvent être tirés de ce travail :

- La combinaison de signaux complémentaires (contenu + similarité utilisateur + similarité item) renforce la robustesse et la qualité des recommandations.
- Les approches hybrides offrent un compromis efficace entre performance et explicabilité, notamment pour gérer le cold-start.
- L'utilisation d'un méta-modèle (stacking) permet de tirer profit des interactions non-linéaires entre les signaux, offrant un gain supplémentaire par rapport à une simple combinaison linéaire.

Enfin, ce projet met en évidence que la conception d'un système de recommandation performant nécessite non seulement le choix judicieux des modèles, mais également un prétraitement rigoureux des données, un réglage fin des hyperparamètres et une évaluation complète des métriques de qualité, de diversité et de couverture. Les perspectives d'amélioration incluent l'intégration de signaux implicites, l'extension à des datasets plus larges et l'optimisation des architectures hybrides pour réduire la complexité computationnelle.