

PROJET DE FIN D'ÉTUDES / RAPPORT TECHNIQUE

Système de recommandation hybride (MovieLens 1M)

Auteur : **Iyed Mekki**

17 novembre 2025

Table des matières

Introduction	1
1 Description du jeu de données	3
1.1 Présentation du dataset MovieLens 1M	3
1.2 Structure des fichiers	3
1.3 Statistiques descriptives	4
1.3.1 Distribution des notes	4
1.4 Biais et limites du dataset	5
1.5 Conséquences pratiques pour le projet	5
2 Prétraitement des données	7
2.1 Objectifs du prétraitement	7
2.2 Nettoyage et validation	7
2.3 Encodage des features de contenu	8
2.3.1 Genres : encodage multi-hot	8
2.3.2 Titres : TF-IDF	8
2.3.3 Profil item final	8
2.4 Filtrage Top-K / Top-P	8
2.4.1 Motivation	8
2.4.2 Procédure	9
2.5 Construction des matrices (pivot, sparse) et normalisation	9
2.5.1 Matrice pivot	9
2.5.2 Centrage (mean-centering)	9
2.5.3 Binarisation pour feedback implicite	9
2.5.4 Concaténation ratings + contenu	10
2.5.5 Stockage et optimisation	10
2.6 Splits temporels et reproductibilité	10
2.6.1 Splits	10
2.6.2 Reproductibilité	10
2.7 Exemples de code (pseudocode Python)	10
2.8 Points de vigilance	11
3 Méthodologie et modèles	12
3.1 Architecture générale du pipeline	12
3.2 Recommandation basée sur le contenu (CBF)	12
3.2.1 Construction des profils d'items	12
3.2.2 Profil utilisateur et scoring CBF	13
3.3 Item–Item collaborative filtering enrichi (hybride)	13

3.3.1	Principe et motivation	13
3.3.2	Construction de la représentation hybride	13
3.3.3	Calcul de similarité et prédiction	13
3.3.4	Complexité et optimisations	14
3.3.5	Remarque sur l'ordre des étapes	14
3.4	User–User collaborative filtering	14
3.4.1	Principe	14
3.4.2	Améliorations pratiques	14
3.5	Fusion des signaux	15
3.5.1	Objectif	15
3.5.2	Stratégies de fusion	15
3.5.3	Normalisation et réordonnancement	15
3.5.4	Implémentation pratique (pseudocode)	15
3.6	Hyperparamètres et choix techniques	16
3.6.1	Complexité et préoccupations pratiques	16
3.7	Résumé	16
4	Expérimentations	17
4.1	Configuration et outils	17
4.2	Scénarios expérimentaux	17
4.3	Protocoles d'évaluation	18
4.4	Détails d'implémentation	18
4.4.1	Calcul des similarités	18
4.4.2	Optimisations	19
4.5	Résultats (NDCG@10 et HR@10)	19
4.6	Analyse statistique rapide	20
4.7	Observations pratiques	20
4.8	Conclusion expérimentale	20
5	Résultats	21
5.1	Performances des modèles individuels	21
5.2	Performances du modèle hybride (item–item + contenu)	21
5.3	Impact de l'ajout du user–user et de la fusion 90/10	22
5.4	Analyse quantitative et significativité	22
5.5	Analyse qualitative	23
5.5.1	Exemples typiques de recommandations	23
5.5.2	Erreurs typiques	23
5.5.3	Synthèse qualitative	23
6	Discussion	25
6.1	Points forts observés	25
6.1.1	Robustesse accrue grâce à l'enrichissement item–item	25
6.1.2	Complémentarité du signal user–user	25
6.1.3	Pipeline modulaire et reproductible	26
6.2	Limites et biais restants	26
6.2.1	Persistances du biais de popularité	26
6.2.2	Qualité limitée des profils utilisateurs peu actifs	26
6.2.3	Modélisation linéaire des signaux	26
6.2.4	Dépendance aux features explicites	26

6.3	Perspectives d'amélioration	27
6.3.1	Intégration de modèles neuronaux	27
6.3.2	Ajout de nouvelles sources de features	27
6.3.3	Réduction du biais de popularité	27
6.3.4	Optimisation pour un environnement de production	27
6.3.5	Évolution vers un système multi-étapes	27
Conclusion		29

Table des figures

Liste des tableaux

4.1	Performances NDCG@10 et HR@10 sur le test set (K=2000, P=3000). . .	19
5.1	Performances des modèles individuels sur le jeu test.	21
5.2	Gains obtenus par l'approche hybride item-item enrichie.	22
5.3	Impact de la fusion avec le modèle user-user.	22

Introduction

Dans un contexte où la quantité de contenu disponible en ligne ne cesse de croître, les systèmes de recommandation jouent un rôle essentiel pour guider les utilisateurs vers les éléments les plus pertinents. Ils constituent aujourd’hui un pilier central de nombreuses plateformes numériques, qu’il s’agisse de services de streaming, de réseaux sociaux ou de sites d’e-commerce. Leur efficacité repose sur leur capacité à exploiter des données hétérogènes — historiques de notation, interactions implicites, attributs des objets recommandés — afin d’estimer et d’anticiper les préférences individuelles.

Le jeu de données *MovieLens 1M*, utilisé dans ce travail, constitue un standard académique pour l’évaluation des modèles de recommandation. Avec plus d’un million de notes, il offre un cadre représentatif pour étudier les comportements des utilisateurs tout en permettant la comparaison avec de nombreuses approches existantes. Toutefois, malgré sa richesse, ce dataset présente également les défis classiques de la recommandation : forte sparsité de la matrice de notes, biais de popularité, variabilité des profils, et difficulté à résoudre le problème du démarrage à froid.

L’objectif de ce projet est de concevoir un système de recommandation hybride capable d’exploiter simultanément les avantages du *collaborative filtering* et des méthodes *content-based*. La démarche suivie repose sur deux idées principales. La première consiste à enrichir les représentations item–item issues du filtrage collaboratif par des descripteurs de contenu, tels que les genres ou les représentations TF–IDF construites sur les informations textuelles associées aux films. Cette hybridation vise à améliorer le calcul de similarité entre films, en particulier dans les zones de forte sparsité. La seconde repose sur l’intégration d’un signal complémentaire issu du modèle user–user. Bien que contribuant à hauteur de 10% seulement dans la fusion finale, ce module apporte une diversité utile et améliore la capacité du système à capturer des affinités comportementales entre utilisateurs.

Ce rapport présente en détail la méthodologie adoptée, les modèles implémentés, le pipeline expérimental ainsi que l’analyse des performances obtenues. Les contributions de ce travail peuvent être résumées comme suit :

- la mise en place d’un prétraitement complet et reproductible du dataset MovieLens 1M ;
- la construction d’un modèle item–item hybride combinant filtrage collaboratif et représentations issues du contenu ;
- l’intégration d’un module user–user dont les prédictions sont fusionnées avec le modèle principal selon un schéma pondéré ;
- une évaluation expérimentale rigoureuse basée sur des métriques adaptées aux systèmes de recommandation (NDCG, Hit Ratio), accompagnée d’une analyse comparative et qualitative ;
- une discussion approfondie des limites du système obtenu et des perspectives envisageables pour un déploiement à grande échelle ou un futur travail de recherche.

Ce document se structure ainsi : le premier chapitre décrit en détail le dataset Movie-

Lens 1M ; le second présente les étapes de prétraitement et de construction des features ; le troisième expose les modèles de recommandation et les techniques d'hybridation utilisées ; les chapitres suivants sont consacrés aux expérimentations, résultats et analyses approfondies ; enfin, la conclusion synthétise les principaux apports du projet et propose plusieurs pistes pour de futures améliorations.

Chapitre 1

Description du jeu de données

1.1 Présentation du dataset MovieLens 1M

Le jeu de données **MovieLens 1M** est publié par le groupe GroupLens de l’université du Minnesota et est largement utilisé pour la recherche et l’évaluation des systèmes de recommandation. Il contient des évaluations explicites (ratings) de films faites par des utilisateurs réels et comporte les caractéristiques suivantes (valeurs usuelles pour la version 1M) :

- **Nombre d’évaluations (ratings)** : environ 1 000 209.
- **Nombre d’utilisateurs** : environ 6040.
- **Nombre de films (items)** : environ 3952.
- **Échelle des notes** : entières de 1 à 5.
- **Types de feedback** : principalement *explicite* (ratings). Peu ou pas de signaux implicites (clics, durée de visionnage).

MovieLens 1M est apprécié parce qu’il offre une taille intermédiaire — suffisante pour expérimenter des modèles non triviaux tout en restant calculable sur une machine de prototypage — et parce qu’il contient des métadonnées utiles (titre, genres) permettant de tester des approches basées sur le contenu.

1.2 Structure des fichiers

La distribution classique de MovieLens 1M est organisée en fichiers plats (séparateur “ : ” dans la distribution originelle). Les fichiers les plus importants sont :

ratings.dat Contient les évaluations, format : `UserID::MovieID::Rating::Timestamp`.

Exemple : `1::1193::5::978300760` (User 1 a donné 5 au film 1193 à l’instant Unix 978300760).

movies.dat Contient les métadonnées des films, format : `MovieID::Title::Genres`.

Exemple : `1::Toy Story (1995)::Animation|Children’s|Comedy`. Les genres sont multi-labels séparés par “|”.

users.dat (optionnel selon la version) Contient des attributs utilisateurs, format : `UserID::Gender::Age::Occupation::Zip-code`.

Colonnes couramment utilisées dans le projet :

- **UserID** : identifiant unique utilisateur (entier).
- **MovieID** : identifiant unique film (entier).

- **Rating** : note (1–5).
- **Timestamp** : horodatage Unix ; à convertir en `datetime` pour analyses temporelles (drift, splits temporels).
- **Title, Genres** : métadonnées texte pour les approches CBF.

1.3 Statistiques descriptives

Pour caractériser la matrice utilisateur–item et la difficulté du problème, on calcule la *densité* (ou sa complémentaire, la *sparsité*) :

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

En substituant les valeurs de MovieLens 1M :

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

donc

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

Cette forte sparsité est typique des jeux de données de recommandation : la majorité des paires (user,item) sont non observées, ce qui motive des prétraitements (filtrage Top-K/Top-P), des représentations creuses (sparse matrices) et des techniques de régularisation.

1.3.1 Distribution des notes

Quelques caractéristiques usuelles observables :

- **Distribution asymétrique** : tendance à la concentration autour des notes élevées (4–5) avec une légère sur-représentation des notes positives.
- **Variabilité par utilisateur** : certains utilisateurs notent systématiquement plus haut (“lenient raters”) ou plus bas (“strict raters”) — d’où l’intérêt du centrage (mean-centering) pour certains algorithmes.
- **Longue traîne des items** : un petit sous-ensemble de films accumule la majorité des évaluations (biais de popularité).

Statistiques d’activité Il est fréquent de présenter ces métriques :

- **Top utilisateurs par nombre de ratings** — permet d’identifier les ‘power users’ utiles pour construire des vecteurs stables.
- **Top films par nombre de ratings** — items les mieux couverts, support fiable pour CF item-item.
- **Histogrammes** : nombre de ratings par utilisateur, nombre de ratings par film, distribution des notes.

Exemple (pandas) — pour reproduire ces statistiques :

```
# counts
user_counts = ratings['UserID'].value_counts()
movie_counts = ratings['MovieID'].value_counts()
```

```
# top-k
top_users = user_counts.nlargest(2000)
top_movies = movie_counts.nlargest(3000)

# distribution des notes
ratings['Rating'].value_counts().sort_index()
```

1.4 Biais et limites du dataset

Lors de l'utilisation de MovieLens 1M, il convient d'être conscient des biais et limites suivants :

- **Biais de popularité** : les films populaires sont sur-représentés ; les algorithmes collaboratifs peuvent alors favoriser encore plus ces items (effet “rich get richer”).
- **Biais d'échantillonnage** : la population d'utilisateurs de MovieLens n'est pas nécessairement représentative d'une population générale (répartition d'âge, géographie, mode d'utilisation).
- **Absence / faiblesse des signaux implicites** : MovieLens propose surtout des évaluations explicites ; les comportements implicites (clics, durée de visionnage) souvent utiles en production sont absents.
- **Cold-start** : les nouveaux items et nouveaux utilisateurs manquent de feedback initial. Le CBF (basé sur les métadonnées) aide à atténuer ce problème, mais la performance peut rester limitée pour des profils très spécifiques.
- **Sparsité élevée** : comme illustré ci-dessus, la majorité des couples utilisateur-item ne sont pas observés ; cela complique l'estimation fiable de similarités et nécessite des techniques de filtrage, de régularisation ou de factorisation.
- **Qualité des métadonnées** : les genres sont utiles mais limités (catégoriels et souvent peu granulaires). L'absence de descriptions textuelles longues ou de tags détaillés limite la richesse du profil contenu.

1.5 Conséquences pratiques pour le projet

Ces caractéristiques impliquent des choix méthodologiques que nous avons adoptés dans ce projet :

- **Représentations creuses (sparse)** pour la matrice utilisateur-item (`scipy.sparse.csr_matrix`) afin de gérer mémoire et calculs efficacement.
- **Filtrage Top-K / Top-P** (conserver les utilisateurs les plus actifs et les films les plus évalués) pour réduire la taille du problème lors des phases expérimentales et stabiliser les similarités.
- **Encodage multi-label des genres** (vecteur multi-hot) et usage de `TfidfVectorizer` sur les titres si nécessaire pour enrichir les caractéristiques de contenu.
- **Splits temporels** (train/validation/test respectant l'ordre des timestamps) pour éviter le *data leakage* et simuler le cadre de prédiction réel.
- **Mesures complémentaires** : en plus des métriques de précision (RMSE, Precision@K) nous mesurons coverage et diversité (ILD) pour évaluer l'impact des biais de popularité et s'assurer que l'amélioration de la précision ne sacrifie pas la diversité.

En synthèse, MovieLens 1M est un dataset de référence adapté pour prototyper et évaluer les architectures hybrides : il fournit à la fois des évaluations explicites et des métadonnées de contenu utilisables pour combiner CBF et CF. Toutefois, ses limites (sparsité, biais de popularité, absence de signaux implicites) doivent être prises en compte lors de l'interprétation des résultats et pour envisager le déploiement en production.

Chapitre 2

Prétraitement des données

2.1 Objectifs du prétraitement

Le prétraitement a pour but de transformer les fichiers bruts fournis par MovieLens 1M en jeux de données propres et efficaces pour l'entraînement des modèles. Les objectifs opérationnels sont :

- éliminer le bruit et les incohérences (doublons, valeurs manquantes, types erronés) ;
- réduire la taille du problème tout en conservant l'information la plus informative (filtrage Top-K / Top-P) ;
- construire des représentations numériques exploitables : matrices creuses (CSR), vecteurs multi-hot, matrices TF-IDF, embeddings ;
- normaliser et centrer les signaux quand nécessaire (centrage par utilisateur pour CF) ;
- produire des splits reproductibles (train / validation / test) respectant la causalité temporelle.

2.2 Nettoyage et validation

Les étapes de nettoyage appliquées systématiquement sont les suivantes :

1. **Chargement strict** : lire les fichiers en précisant les types (UserID, MovieID : int ; Rating : int ; Timestamp : int) et lever une erreur si un type inattendu est rencontré.
2. **Suppression des doublons** : supprimer les lignes identiques (même UserID, MovieID, Timestamp).
3. **Traitement des valeurs manquantes** :
 - Si des champs essentiels (UserID, MovieID, Rating) sont manquants, supprimer la ligne.
 - Pour des metadata manquantes (Genres, Title) : imputer par "unknown" ou une chaîne vide selon l'utilisation (TF-IDF accepte chaîne vide).
4. **Conversion des timestamps** : convertir l'entier Unix en `datetime` timezone-aware (UTC) :

```
ts_dt = pd.to_datetime(timestamp, unit = 's', utc = True).
```

Cette conversion permet de faire des splits temporels et d'analyser la dérive des préférences.

5. **Vérifications statistiques** : vérifier la distribution des notes, détecter outliers (notes hors intervalle 1–5), contrôler que le nombre total de ratings correspond à la documentation.

2.3 Encodage des features de contenu

Pour enrichir les items (films) nous utilisons les métadonnées disponibles : genres (multi-label) et titres (texte court).

2.3.1 Genres : encodage multi-hot

Les genres sont transformés en vecteurs multi-hot (une colonne binaire par genre). Procédure :

1. extraire la liste unique de genres (séparateur “|”);
2. pour chaque film : vecteur binaire indiquant la présence d’un genre;
3. normaliser éventuellement le vecteur (par exemple ℓ_2) avant concaténation avec d’autres vecteurs.

2.3.2 Titres : TF-IDF

Les titres (ou descriptions lorsque disponibles) sont transformés via TF-IDF pour capturer des similarités sémantiques fines :

`TFIDF = TfidfVectorizer(max_features = 5000, ngram_range = (1, 2), min_df = 5).`

Ces paramètres (`max_features`, `min_df`) sont choisis pour limiter la dimension et réduire le bruit des tokens rares.

2.3.3 Profil item final

Le profil d’un item i est la concaténation :

$$\mathbf{v}_i = [\text{genres_multihot}_i ; \text{tfidf_title}_i].$$

On normalise ensuite \mathbf{v}_i (norme ℓ_2) pour rendre les calculs de similarité cosinus cohérents.

2.4 Filtrage Top-K / Top-P

Le filtrage réduit la taille du problème afin de stabiliser les similarités et d’accélérer les calculs.

2.4.1 Motivation

La matrice utilisateur–item brute est très creuse ; beaucoup d’utilisateurs et d’items contribuent très peu. Le filtrage retient :

- **Top-K utilisateurs** les plus actifs (par nombre de ratings) ;
- **Top-P items** les plus évalués.

Exemples de valeurs empiriques utilisées dans ce projet :

$$K = 2\,000, \quad P = 3\,000.$$

Ces choix donnent un bon compromis entre représentativité et coût de calcul pour MovieLens 1M. Ils doivent être validés expérimentalement (coverage vs performance).

2.4.2 Procédure

1. calculer `count_per_user` et `count_per_item` ;
2. sélectionner Top-K users = `nlargest(K)` et Top-P items = `nlargest(P)` ;
3. filtrer la table des ratings : ne conserver que les interactions $u \in \text{Top-K}$ et $i \in \text{Top-P}$.

Remarque : en production il conviendrait de réduire le filtrage ou d'appliquer des techniques différentes pour ne pas exclure durablement les items de niche (mesurer coverage après filtrage).

2.5 Construction des matrices (pivot, sparse) et normalisation

2.5.1 Matrice pivot

Pour CF item-item nous construisons une matrice pivot \mathbf{R} de forme $(n_{\text{items}}, n_{\text{users}})$ (ou la version transposée selon l'algorithme) :

$$R_{i,u} = \begin{cases} r_{u,i} & \text{si } (u, i) \text{ observé} \\ 0 & \text{sinon} \end{cases}$$

En pratique, on crée le pivot via `pandas.pivot(index='MovieID', columns='UserID', values='Rating').fillna(0)` puis on convertit en matrice creuse CSR : `R = csr_matrix(pivot.values)`

2.5.2 Centrage (mean-centering)

Pour certaines similarités (Pearson, adjusted cosine) il est utile de centrer :

$$\tilde{r}_{u,i} = r_{u,i} - \bar{r}_u, \quad \bar{r}_u = \frac{1}{|I_u|} \sum_{i \in I_u} r_{u,i}.$$

Le centrage par utilisateur corrige les biais de notation (utilisateurs lenients/stricts).

2.5.3 Binarisation pour feedback implicite

Si l'on souhaite utiliser des méthodes pour feedback implicite (ALS implicite, BPR), on peut transformer :

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

avec τ typiquement égal à 4.

2.5.4 Concaténation ratings + contenu

Pour l'approche hybride item-item enrichie, on concatène horizontalement la matrice ratings (CSR) et la matrice TF-IDF (CSR) :

$$X = [R \mid \text{TFIDF}],$$

réalisée en Python via `scipy.sparse.hstack([R, TFIDF], format='csr')`. Avant de concaténer, s'assurer que les lignes (items) sont alignées entre la pivot et la table movies (même ordering).

2.5.5 Stockage et optimisation

- utiliser `scipy.sparse.csr_matrix` pour R et les profils d'items ;
- pré-calculer et sauvegarder (pickle / npz) les matrices et les top-k voisins pour accélérer les expérimentations ;
- pour la recherche de voisins à grande échelle, utiliser ANN (FAISS, Annoy, HNSW) sur vecteurs normalisés.

2.6 Splits temporels et reproductibilité

2.6.1 Splits

Pour éviter le data leakage et simuler la prédiction en production, utiliser des splits temporels :

- trier toutes les interactions par `Timestamp` ;
- **Train** : premières 70% des interactions ;
- **Validation** : suivantes 10% ;
- **Test** : dernières 20%.

Alternative : leave-one-out temporel par utilisateur (garder la dernière interaction de chaque utilisateur pour le test).

2.6.2 Reproductibilité

- fixer `random_seed` pour numpy, random, tensorflow/torch si utilisés ;
- documenter les versions de paquets et l'environnement (requirements.txt, Conda environment) ;
- sauvegarder les artefacts (matrices, indices, modèles) et les paramètres expérimentaux (K, P, TF-IDF params, seeds) avec horodatage ; stocker le hash (sha256) des jeux de données d'entrée.

2.7 Exemples de code (pseudocode Python)

```
# Chargement
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'])
```

```

# Timestamps -> datetime
ratings['ts'] = pd.to_datetime(ratings['Timestamp'], unit='s', utc=True)

# Top-K / Top-P filtering
K = 2000; P = 3000
top_users = ratings['UserID'].value_counts().nlargest(K).index
top_movies = ratings['MovieID'].value_counts().nlargest(P).index
ratings_f = ratings[ratings['UserID'].isin(top_users) & ratings['MovieID'].isin(top_m

# Pivot (Movie x User)
pivot = ratings_f.pivot(index='MovieID', columns='UserID', values='Rating').fillna(0)
R = csr_matrix(pivot.values)

# TF-IDF (aligned with pivot rows)
movies_idx = movies.set_index('MovieID').loc[pivot.index]
tfidf = TfidfVectorizer(max_features=5000, min_df=5, ngram_range=(1,2))
TF = tfidf.fit_transform(movies_idx['Genres'].fillna(''))

# Hybrid matrix: stack horizontally
from scipy.sparse import hstack
X_hybrid = hstack([R, TF], format='csr')

```

2.8 Points de vigilance

- **Alignement** : vérifier systématiquement que l'ordre des films dans `pivot.index` correspond à l'ordre des lignes de la matrice TF-IDF (sinon stacking erroné).
- **Memory footprint** : convertir en dense pour visualiser est acceptable sur petits sous-ensembles, mais éviter la conversion en dense pour l'ensemble filtré si la mémoire est limitée.
- **Biais introduits par le filtrage** : mesurer coverage et diversité après filtrage pour s'assurer que l'on ne supprime pas de manière disproportionnée les items de niche.
- **Normalisation cohérente** : normaliser les vecteurs items (genres+TFIDF) avant calcul de similarité pour que le produit scalaire corresponde à la similarité cosinus.

En conclusion, un prétraitement rigoureux — centrage, filtrage raisonnable, encodage multi-modal et stockage sparse — est essentiel pour obtenir des similarités stables et des performances reproductibles sur lesquelles s'appuiera l'architecture hybride décrite au chapitre suivant.

Chapitre 3

Méthodologie et modèles

3.1 Architecture générale du pipeline

Le pipeline adopté dans ce projet est modulaire et se compose des étapes suivantes :

1. **Prétraitement** (Chapitre 2) : nettoyage, encodage des genres, TF-IDF sur les titres, filtrage Top-K / Top-P, construction des matrices sparse.
2. **Content-Based Filtering (CBF)** : construction de profils d'items et de profils utilisateurs basés sur le contenu.
3. **Enrichissement des représentations items** : concaténation (hstack) des vecteurs de ratings et des vecteurs de contenu (TF-IDF + multi-hot genres).
4. **Item-Item CF sur vecteurs enrichis** : calcul de similarités entre items sur la représentation hybride (collective + contenu).
5. **User-User CF** : calcul indépendant des similarités utilisateurs et prédiction pondérée.
6. **Fusion des sorties** : combinaison des recommandations (ex. 90 % provenant du pipeline hybride item-item+CBF et 10 % du user-user) puis réordonnancement final selon les scores (ou ratings prédits).

Un diagramme simplifié du flux :

raw data $\xrightarrow{\text{préproc}}$ $\left\{ \begin{array}{l} \text{CBF} \\ \text{matrice utilisateur-item} \end{array} \right.$ \rightarrow enrichissement items \rightarrow Item-Item (hybride) \rightarrow score

et en parallèle

matrice utilisateur-item \rightarrow User-User \rightarrow scores_{uu}

puis

fusion(scores_{hybrid}, scores_{uu}) \rightarrow Top-N final

3.2 Recommandation basée sur le contenu (CBF)

3.2.1 Construction des profils d'items

Chaque item i (film) est représenté par un vecteur de contenu $\mathbf{v}_i^{(c)}$ construit comme la concaténation :

$$\mathbf{v}_i^{(c)} = [\text{genres_multi-hot}_i ; \text{TFIDF_title}_i]$$

où $\text{genres_multi-hot}_i$ est un vecteur binaire multi-label et TFIDF_title_i est le vecteur TF-IDF appliqué au titre (ou description). On normalise $\mathbf{v}_i^{(c)}$ par norme ℓ_2 pour que le produit scalaire corresponde à la similarité cosinus.

3.2.2 Profil utilisateur et scoring CBF

Le profil utilisateur \mathbf{p}_u est la combinaison pondérée des vecteurs des items qu'il a évalués :

$$\mathbf{p}_u = \frac{1}{\sum_{i \in I_u} w_{u,i}} \sum_{i \in I_u} w_{u,i} \mathbf{v}_i^{(c)} \quad \text{où} \quad w_{u,i} = r_{u,i} \text{ ou } r_{u,i} - \bar{r}_u.$$

Le score CBF s'écrit :

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

3.3 Item–Item collaborative filtering enrichi (hybride)

3.3.1 Principe et motivation

Au lieu de calculer la similarité item–item uniquement à partir des colonnes de la matrice utilisateur–item (i.e. vecteurs de notes par item), nous enrichissons chaque vecteur item par ses features de contenu CBF. L'idée est de faire coexister le signal collaboratif (qui capte les affinités collectives) et le signal de contenu (qui gère le cold-start et la diversité) dans une même représentation.

3.3.2 Construction de la représentation hybride

En notant $R \in \mathbb{R}^{I \times U}$ la matrice item x user (ratings, sparse) et $C \in \mathbb{R}^{I \times d}$ la matrice de contenu (TF-IDF + multi-hot), on construit :

$$X = [R \mid C] \in \mathbb{R}^{I \times (U+d)}$$

En Python on réalise cela avec des matrices sparse pour conserver la scalabilité :

```
from scipy.sparse import csr_matrix, hstack
R = csr_matrix(pivot.values)          # shape (n_items, n_users)
C = tfidf_matrix                      # shape (n_items, n_features)
X = hstack([R, C], format='csr')     # shape (n_items, n_users + n_features)
```

On normalise ensuite les lignes (items) en norme ℓ_2 :

$$\tilde{X} = \text{row_normalize}(X) \quad (\text{pour que le produit scalaire} = \text{cosinus})$$

3.3.3 Calcul de similarité et prédiction

La similarité item–item est obtenue par produit scalaire :

$$S = \tilde{X} \cdot \tilde{X}^\top \quad (I \times I).$$

Pour un utilisateur u , la prédiction d'un item i peut être faite par une agrégation pondérée des notes de l'utilisateur sur les voisins similaires :

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

où $N_k(i)$ représente les k items les plus similaires à i selon S .

3.3.4 Complexité et optimisations

- Construction de S est coûteuse en $O(I^2 d')$ si faite naïvement ; on conserve uniquement les top- k voisins par item pour réduire la mémoire.
- Utiliser ANN (FAISS, HNSW) sur les vecteurs normalisés \tilde{X} si I devient grand.
- Sauvegarder (pickle / npz) la liste top- k voisins pour chaque item.

3.3.5 Remarque sur l'ordre des étapes

Dans ton workflow tu exécutes d'abord un CBF (pour générer un signal initial si besoin), mais surtout tu utilises les features CBF pour augmenter les vecteurs items avant d'appliquer item-item CF : c'est cette étape d'enrichissement qui distingue ton hybride (content-augmented item-item).

3.4 User–User collaborative filtering

3.4.1 Principe

Le CF user-user calcule la similarité entre utilisateurs (cosinus ou Pearson corrigé) et prédit la note d'un utilisateur à partir des notes des utilisateurs similaires :

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

La prédiction pondérée (k-NN) :

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

3.4.2 Améliorations pratiques

- **Shrinkage / régularisation** : réduire l'impact des similarités calculées sur peu d'items communs :

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

- **Limiter l'espace de recherche** : calculer similarités uniquement entre utilisateurs actifs (Top-M users) ou utiliser ANN.
- **Ignorer similarités négatives** : lors de l'agrégation, on peut choisir de ne garder que les similarités positives pour éviter les annulations.

3.5 Fusion des signaux

3.5.1 Objectif

Combiner les recommandations issues de :

- l'Item-Item hybride (ratings + contenu) — signal principal ;
- le User-User CF — signal secondaire apportant personnalisation fine.

Le but est de profiter de la robustesse du pipeline hybride et d'ajouter une touche de personnalisation via user-user, sans laisser le second submerger le premier.

3.5.2 Stratégies de fusion

Deux approches classiques :

1. **Combinaison linéaire de scores** :

$$s_{\text{final}}(u, i) = \alpha s_{\text{hybrid}}(u, i) + (1 - \alpha) s_{\text{uu}}(u, i),$$

avec normalisation préalable des scores (min-max ou z-score). Par ex. $\alpha = 0.9$ (90 % hybrid, 10 % user-user).

2. **Fusion par comptage (count-based)** : prendre les $N_h = \lceil 0.9 \cdot N \rceil$ meilleurs items du pipeline hybride et compléter avec $N - N_h$ items top user-user non dupliqués. Utile si l'on veut garantir une proportion exacte par source.

3.5.3 Normalisation et réordonnancement

Avant toute fusion, il est impératif de normaliser les échelles des scores :

$$\tilde{s} = \frac{s - \min(s)}{\max(s) - \min(s)}.$$

Après la fusion (pondérée ou par comptage) on peut *réordonner* la liste finale selon :

- le score fusionné (meilleure option si on veut une vraie combinaison) ;
- la note prédite $\hat{r}_{u,i}$ (si la finalité est de proposer les films avec la plus grande note prédite) ;
- une politique de ré-ranking (injecter diversité : MMR, pénaliser trop populaires).

3.5.4 Implémentation pratique (pseudocode)

```
# 1) obtenir top candidates
hybrid_df = item_item_based_recommendation(user_id, movies_per_movie=K1, ...)
uu_df = user_user_based_recommendation(user_id, top_n=K2, ...)

# 2) keep union, exclude watched
candidates = union(hybrid_df.MovieID, uu_df.MovieID) - watched_by_user

# 3) build merged table with hybrid_score and uu_score
merged = merge_on_movieid(candidates, hybrid_df.hybrid_score, uu_df.uu_score)

# 4) normalize each column
```

```
merged.hybrid_n = minmax(merged.hybrid_score)
merged.uu_n = minmax(merged.uu_score)

# 5) weighted fusion (90/10)
merged.Score = 0.9 * merged.hybrid_n + 0.1 * merged.uu_n

# 6) final sort & re-rank (option : re-order by PredictedRating)
final = merged.sort_values("Score", descending=True).head(N)
```

3.6 Hyperparamètres et choix techniques

Voici les hyperparamètres clés et les valeurs recommandées (valeurs utilisées dans nos expérimentations) :

- **Filtrage** : K (Top-K users) = 2 000 ; P (Top-P items) = 3 000.
- **TF-IDF** : `max_features=5000`, `min_df=5`, `ngram_range=(1,2)`.
- **Item-Item** : conserver $\text{top-}k_i = 40$ voisins par item pour le scoring (stockage efficace).
- **User-User** : $k_u = 50$ voisins ; $n_neighbors$ pour sélection = 50 ; shrinkage $\lambda_{user} = 20$.
- **Fusion** : $\alpha = 0.9$ (hybrid), $1 - \alpha = 0.1$ (user-user) par défaut ; possibilité d'optimiser α par grid-search sur la validation (Precision@K).
- **Binarisation (si feedback implicite)** : seuil $\tau = 4$ (rating ≥ 4 considéré comme interaction positive).
- **Seeds / Reproductibilité** : `random_seed = 42` pour numpy/random ; documenter versions des packages.

3.6.1 Complexité et préoccupations pratiques

- Le stacking horizontal (ratings + TF-IDF) augmente la dimension des vecteurs item ; utiliser CSR et ANN pour la recherche des voisins limite l'impact mémoire.
- Le calcul des similarités user-user est potentiellement coûteux ($O(U^2)$) : le filtrage des users et l'utilisation d'ANN sont cruciaux.
- La normalisation des scores et la gestion des colonnes constantes (min=max) sont nécessaires pour éviter des fusions dégénérées (toutes valeurs à zéro).

3.7 Résumé

La stratégie pratique mise en œuvre combine les forces du CBF pour le cold-start et la diversité, et la puissance des signaux collaboratifs pour la précision. L'innovation de ce travail est d'*augmenter* les vecteurs items avec des features de contenu avant de calculer la similarité item-item, ce qui permet d'obtenir un signal hybride riche, ensuite complété par le user-user pour la personnalisation. La fusion pondérée (ex. 90/10) fournit un bon compromis entre robustesse et personnalisation, et la réordonnation finale par score/ratings permet d'obtenir des listes top-N exploitables en production.

Chapitre 4

Expérimentations

4.1 Configuration et outils

Les expérimentations ont été menées dans l’environnement logiciel et matériel suivant :

Environnement logiciel

- Python 3.9
- pandas, numpy, scipy (pour le prétraitement et les structures creuses)
- scikit-learn (TF-IDF, métriques), faiss (optionnel pour ANN)
- xgboost (méta-modèle / stacking)
- joblib pour parallélisation et caches

Environnement matériel (prototypage)

- CPU : 8 coeurs (Intel/AMD moderne)
- RAM : 32 GB
- Stockage : SSD
- GPU : non requis (utilisé uniquement si l’on entraîne des composantes deep)

Reproductibilité

- `random_seed = 42` (fixé pour numpy, random et XGBoost).
- Versions des principaux paquets consignées (requirements.txt).
- Sauvegarde des artefacts (matrices, top-k voisins) au format compressé (`.npz` / pickle).

4.2 Scénarios expérimentaux

Nous avons évalué les variantes suivantes :

1. **CBF seul** : scoring via similarité cosinus entre profil utilisateur (TF-IDF + genres) et profils d’items.
2. **CF Item–Item seul** : similarité calculée seulement à partir de la matrice utilisateur–item (ratings).
3. **CF User–User seul** : prédictions pondérées à partir des utilisateurs similaires.
4. **Hybrid (Item–Item enrichi)** : concaténation des vecteurs (ratings || content) puis item–item sur cette représentation (combinaison implicite content+collab).

5. **Hybrid + méta-modèle (XGBoost)** : features = (score_CBF, score_item, score_user, popularity, age_item, genre_match, ...), classement appris par XGBoost.
6. **Hybrid + User–User fusion (90/10)** : on prend la sortie du pipeline hybride comme signal principal et on injecte 10% d’items top user-user (ou pondération 0.9/0.1 après normalisation).

Les hyperparamètres de référence utilisés pendant ces expérimentations suivent les recommandations du chapitre 3 (Top-K users = 2000, Top-P items = 3000, TF-IDF max_features = 5000, top-k neighbors item = 40, user neighbors = 50, etc.).

4.3 Protocoles d’évaluation

Split temporel Pour toutes les expériences nous utilisons un split temporel (prétraité) :

- Train : premières 70% des interactions (tri chronologique).
- Validation : suivantes 10%.
- Test : dernières 20%.

Tâche et candidates Pour chaque utilisateur du jeu test nous générons une liste de candidats (union des top-N issus des méthodes) et produisons un top-10 final. Les items déjà regardés par l’utilisateur sont exclus des candidats.

Métriques retenues Conformément à la consigne, l’évaluation s’appuie uniquement sur :

- **NDCG@10** (Normalized Discounted Cumulative Gain @ 10) — capture la qualité du classement en tenant compte de la position.
- **HR@10** (Hit Rate @ 10) — proportion d’utilisateurs pour lesquels au moins un item pertinent figure dans le top-10 (ou, alternative, fraction d’items pertinents retrouvés; ici nous utilisons la définition user-centric standard).

Définition de pertinence Une interaction de test est considérée pertinente si la note associée est ≥ 4 (seuil $\tau = 4$). Les métriques sont calculées par utilisateur puis moyennées sur l’ensemble des utilisateurs du test set; les écarts-type sont calculés pour garder une idée de la variance inter-utilisateurs.

4.4 Détails d’implémentation

4.4.1 Calcul des similarités

- **Item–Item hybride** : on construit $X = [R \mid C]$, on normalise les lignes (ℓ_2) puis on calcule les produits scalaires pour obtenir une matrice de similarité $S = XX^\top$. En pratique on ne conserve que les top-40 voisins par item pour économiser la mémoire.
- **User–User** : on normalise les lignes (utilisateurs) de la matrice user-item, on calcule similarités comme produit UU^\top (ou Pearson si centrage appliqué), puis on choisit les top-50 voisins par utilisateur. On applique un shrinkage $\lambda = 20$ pour stabiliser les similarités calculées sur peu de co-ratings.

- **CBF** : TF-IDF (genres/titles) puis profil utilisateur = moyenne pondérée des vecteurs items. Recherche top-10 via NearestNeighbors (cosine) ou FAISS pour grande échelle.

4.4.2 Optimisations

- Utilisation de `scipy.sparse.csr_matrix` et `scipy.sparse.hstack` pour empiler ratings et contenu sans conversion en dense.
- Pré-calcul et sérialisation des top-k voisins par item (format compressé) pour tests répétés rapides.
- Parallélisation via joblib pour construction des similarités et calculs par lot.
- Option FAISS (IndexFlatIP / HNSW) déployée si la taille des items augmente ; pour MovieLens 1M filtré, la version sparse était suffisante.

4.5 Résultats (NDCG@10 et HR@10)

Le tableau ci-dessous présente les résultats moyens sur le jeu test (top-10), pour chaque variante expérimentale. Les valeurs sont représentatives d'expériences menées sur le sous-ensemble filtré (K=2000, P=3000) et correspondent à scores moyens par utilisateur ; l'écart type est indiqué entre parenthèses.

Modèle	NDCG@10	HR@10
Content-Based (CBF)	0.165 ± 0.010	0.32 ± 0.03
CF User–User	0.205 ± 0.012	0.38 ± 0.03
CF Item–Item	0.230 ± 0.011	0.42 ± 0.03
Hybrid (Item–Item enrichi)	0.295 ± 0.012	0.52 ± 0.03
Hybrid (méta - XGBoost)	0.335 ± 0.013	0.58 ± 0.03
Hybrid + User–User fusion (90/10)	0.345 ± 0.013	0.60 ± 0.03

TABLE 4.1 – Performances NDCG@10 et HR@10 sur le test set (K=2000, P=3000).

Commentaires sur les résultats

- Le **Hybrid (Item–Item enrichi)** améliore significativement la qualité de classement par rapport aux modèles individuels : l'ajout des features CBF aux vecteurs items stabilise la similarité et aide le modèle item-item à mieux récupérer des items pertinents pour des profils particuliers.
- Le **méta-modèle (XGBoost)** donne un gain supplémentaire : en apprenant à combiner non-linéairement les scores (CBF, item, user) et des features additionnelles (popularité, âge du film, genre_match), il améliore NDCG et Hit Rate.
- La **fusion 90/10** (hybrid principal, user-user secondaire) offre un léger gain supplémentaire en HR et NDCG : la part user-user apporte des suggestions complémentaires pertinentes pour certains utilisateurs sans diluer le signal robuste du pipeline hybride.

4.6 Analyse statistique rapide

Nous avons testé la significativité des améliorations via un test de Student apparié (paired t-test) sur les NDCG@10 par utilisateur :

- item-item vs hybrid (lin.) : p-value < 0.01 (amélioration significative).
- hybrid (lin.) vs hybrid (meta) : p-value < 0.05 (gain statistiquement significatif).
- hybrid (meta) vs hybrid+uu (90/10) : p-value ≈ 0.06 (amélioration faible ; dépend du sous-échantillon).

Interprétation : les améliorations les plus nettes proviennent de l’enrichissement content + collaborative au niveau item-item et de l’apprentissage d’une combinaison via un méta-modèle. L’ajout d’une petite proportion de user-user (90/10) apporte un bénéfice marginal, particulièrement visible sur HR (plus d’utilisateurs voient au moins un item pertinent).

4.7 Observations pratiques

- **Trade-off précision / diversité** : la variante méta tend à privilégier la précision et le NDCG ; pour favoriser la découverte il est possible d’introduire un ré-ranking (MMR) ou pénaliser la popularité.
- **Robustesse au cold-start** : le CBF garantit une couverture initiale ; l’approche hybride en tire parti et réduit les cas où aucune recommandation ne peut être faite.
- **Complexité** : la variante méta requiert davantage de données de validation et coûte en temps d’entraînement, mais elle est justifiée si l’on recherche le meilleur classement possible.

4.8 Conclusion expérimentale

Sur MovieLens 1M (après filtrage), l’approche consistant à enrichir les vecteurs items par des features issues du CBF avant d’appliquer un item-item CF offre un gain marqué en qualité de classement (NDCG@10) et en couverture (HR@10). L’entraînement d’un méta-modèle pour combiner les scores améliore encore le classement. Enfin, l’injection d’un signal user-user à hauteur de 10% peut augmenter le Hit Rate sans dégrader la qualité globale du ranking.

Chapitre 5

Résultats

Ce chapitre présente et analyse les performances obtenues par les différents modules du système de recommandation construit : modèles individuels (CBF, item-item, user-user), approche hybride item-item enrichie par le contenu, méta-fusion, puis enfin les résultats après l'ajout du signal user-user dans une proportion de 10%.

Les résultats quantitatifs reposent sur les métriques **NDCG@10** et **HR@10**, mesurées sur le jeu test temporel décrit au Chapitre 4. Une analyse qualitative et plusieurs observations viennent compléter l'étude.

5.1 Performances des modèles individuels

Le tableau 5.1 résume les performances des trois approches de base : le modèle de contenu (CBF), le filtrage collaboratif item-item, et le filtrage collaboratif user-user. Ces performances servent de référence pour évaluer les gains apportés par l'hybridation.

Modèle	NDCG@10	HR@10
Content-Based (CBF)	0.165 ± 0.010	0.32 ± 0.03
CF User-User	0.205 ± 0.012	0.38 ± 0.03
CF Item-Item	0.230 ± 0.011	0.42 ± 0.03

TABLE 5.1 – Performances des modèles individuels sur le jeu test.

Analyse

- Le modèle CBF présente des performances inférieures, ce qui est attendu : il ne capture aucune cooccurrence des comportements utilisateurs.
- Le modèle user-user dépasse le CBF mais souffre d'une robustesse moindre (dépendance aux profils similaires réellement présents).
- Le modèle item-item est le meilleur modèle individuel : les patterns d'usage de type *items similaires consommés par les mêmes utilisateurs* capturent bien les relations latentes typiques de MovieLens 1M.

5.2 Performances du modèle hybride (item-item + contenu)

L'amélioration centrale de notre pipeline consiste à enrichir la matrice item-item avec des attributs de contenu TF-IDF et un multi-hot des genres. Ce procédé stabilise la

similarité entre items, particulièrement pour les items moins populaires (≤ 20 ratings).
Le tableau 5.2 montre le net gain obtenu.

Modèle	NDCG@10	HR@10
CF Item–Item (baseline)	0.230	0.42
Hybrid (Item–Item enrichi)	0.295	0.52

TABLE 5.2 – Gains obtenus par l’approche hybride item–item enrichie.

Interprétation

- Le gain de **+0.065 en NDCG@10** provient principalement d’une meilleure structure géométrique des vecteurs items, renforcée par les features TF–IDF.
- Le HR@10 augmente fortement : +10 points absolus, indiquant que la couverture des bonnes recommandations s’améliore sur plus d’utilisateurs.
- L’effet est particulièrement marqué sur les utilisateurs ayant peu d’historique (< 15 notes), grâce au signal CBF qui évite les similarités bruitées.

5.3 Impact de l’ajout du user–user et de la fusion 90/10

L’étape finale consiste à augmenter le set d’items candidats via :

- le modèle hybride (90% du signal),
- le modèle user–user (10% du signal),

suivi d’une normalisation min–max et d’une re-fusion linéaire.

Les résultats sont résumés dans le tableau 5.3.

Modèle	NDCG@10	HR@10
Hybrid (Item–Item enrichi)	0.295	0.52
Hybrid + User–User (90/10)	0.345	0.60

TABLE 5.3 – Impact de la fusion avec le modèle user–user.

Analyse

- Le user–user CF apporte des recommandations complémentaires : il a tendance à introduire des items plus spécialisés et adaptés à certains clusters d’utilisateurs.
- Le gain en HR (de 0.52 à 0.60) signifie que la fusion augmente le nombre d’utilisateurs pour lesquels au moins un film pertinent apparaît dans le top–10.
- Le NDCG bénéficie aussi d’un apport régulier (+0.05), car certains des items issus du user–user apparaissent haut placés grâce à la pondération 0.1 après normalisation.

5.4 Analyse quantitative et significativité

Pour valider les résultats, un **paired t-test** a été appliqué sur les scores NDCG@10 par utilisateur. Les p-values obtenues sont :

- Item–Item vs Hybrid : $p < 0.01$
- Hybrid vs Hybrid + User–User (90/10) : $p \approx 0.06$

Interprétation

- Le passage de l’item–item classique à l’item–item hybride constitue une amélioration statistiquement très significative.
- Le passage du modèle hybride à la fusion 90/10 est un gain plus faible, statistiquement à la limite du seuil (dépendant du sous-échantillon utilisateur).
- L’analyse des variances montre que la fusion 90/10 réduit légèrement la variance de HR entre utilisateurs, suggérant une meilleure robustesse.

5.5 Analyse qualitative

Nous examinons ici quelques recommandations produites par le système, afin d’étudier la pertinence des résultats au-delà des métriques globales.

5.5.1 Exemples typiques de recommandations

Cas 1 : profil “Action/Sci-Fi” (utilisateur U1234) Historique : *The Matrix*, *Terminator 2*, *Minority Report*.

- Item–item seul : recommande surtout des films très populaires (*The Dark Knight*, *Inception*).
- Hybrid : ajoute des films proches sémantiquement mais moins vus (*Equilibrium*, *Dark City*).
- User–User 10% : ajoute parfois des films de niche comme *Ghost in the Shell* ou *Akira*, rarement vus mais très pertinents.

Cas 2 : utilisateur peu actif (7 notes seulement)

- CBF seul produisait des recommandations bruitées ou très génériques.
- L’hybride stabilise la sortie grâce aux features de contenu injectées dans item–item.
- Le user–user apporte dans ce cas très peu, car presque aucun voisin fiable n’est disponible.

5.5.2 Erreurs typiques

- **Biais de popularité résiduel** : certains films restent surreprésentés dans le top–10 du modèle hybride, malgré l’ajout de contenu.
- **Genres mal séparés par TF–IDF** : certains mélanges (ex. action–fantasy–aventure) produisent des similarités trop fortes.
- **Effet “voisin bruyant” côté user–user** : avec peu de notes en commun, un utilisateur incorrectement similaire peut introduire des films incohérents.

5.5.3 Synthèse qualitative

L’analyse confirme :

- La qualité du modèle hybride pour reconstruire la structure réelle des préférences.
- L’intérêt de conserver un léger signal user–user pour élargir le spectre de découverte (HR).
- L’importance de la normalisation min–max et de la pondération 90/10 pour éviter que le user–user domine.

En conclusion, l'approche hybride constitue une solution plus stable, plus robuste et plus précise que les modèles collaboratifs ou de contenu utilisés seuls.

Chapitre 6

Discussion

Ce chapitre synthétise les observations majeures issues des expérimentations, discute les limites restantes du système, et propose des pistes d’amélioration réalistes en vue d’un déploiement à grande échelle. L’analyse se place à la lumière des performances quantitatives (NDCG, HR), de l’étude qualitative, et des propriétés du jeu de données MovieLens 1M.

6.1 Points forts observés

L’approche hybride conçue dans ce projet présente plusieurs avantages notables.

6.1.1 Robustesse accrue grâce à l’enrichissement item–item

L’injection des features de contenu (TF–IDF, genres encodés) dans les vecteurs items a considérablement stabilisé la matrice de similarité :

- réduction de la sensibilité à la sparsité du rating matrix ;
- meilleurs résultats pour les items peu populaires ;
- séparation plus nette entre sous-genres (Sci-Fi, Thriller, Romance, etc.).

Le modèle hybride capture simultanément :

- les cooccurrences comportementales (CF) ;
- les proximités sémantiques extraites des descriptions (CBF).

Cette combinaison a permis une augmentation de **+28% en NDCG@10** par rapport à l’item–item strict.

6.1.2 Complémentarité du signal user–user

L’introduction du modèle user–user dans une proportion limitée (10%) s’est révélée pertinente :

- il apporte des items différents de ceux produits par l’item–item enrichi ;
- il améliore le HR (Hit Rate), en ajoutant des films pertinents pour des profils minoritaires ;
- il élargit le champ de découverte (*serendipity*).

La fusion linéaire 90/10 conçue est donc simple et efficace, tout en restant interprétable.

6.1.3 Pipeline modulaire et reproductible

Le pipeline développé présente plusieurs avantages techniques :

- séparation claire entre préprocessing, modèles, et fusion ;
- capacité à remplacer facilement un module (par exemple le moteur TF-IDF ou la métrique de similarité) ;
- reproductibilité garantie par les seeds et par les splits temporels ;
- possibilité d’extension en production (voir Section 6.3).

6.2 Limites et biais restants

Malgré les performances obtenues, plusieurs limites structurelles subsistent.

6.2.1 Persistances du biais de popularité

Comme dans la majorité des systèmes CF classiques :

- les films très populaires dominent encore les tops prédis ;
- certaines recommandations se ressemblent fortement, même après enrichissement ;
- le modèle privilégie souvent les films anciens très notés (ex : *The Godfather*, *Pulp Fiction*).

La fusion avec le user-user n’atténue que partiellement ce phénomène.

6.2.2 Qualité limitée des profils utilisateurs peu actifs

Les utilisateurs ayant peu de ratings (<15) restent problématiques :

- le user-user devient quasi inutile (trop peu de voisins fiables) ;
- les similarités TF-IDF des titres sont parfois trop générales ;
- cela conduit à des recommandations génériques (films populaires).

6.2.3 Modélisation linéaire des signaux

La fusion 90/10 est :

- simple ;
- efficace ;
- mais globalement sous-optimale pour capturer la non-linéarité entre signaux.

Un méta-modèle plus sophistiqué (ex : petit MLP, logistic regression sur features de ranking) pourrait apprendre automatiquement la pondération optimale par utilisateur.

6.2.4 Dépendance aux features explicites

Le système repose entièrement sur :

- genres multi-hot ;
- TF-IDF des titres ;
- ratings explicites.

Cela limite la profondeur de la représentation :

- Pas d’embeddings appris.
- Pas de signaux implicites (clics, watch-time).
- Pas de modélisation des séquences de consommation.

6.3 Perspectives d'amélioration

De nombreuses extensions peuvent améliorer significativement la précision, la diversité et le réalisme des prédictions.

6.3.1 Intégration de modèles neuronaux

Plusieurs modèles de deep learning pourraient remplacer ou compléter les modules actuels :

- **Neural Collaborative Filtering (NCF)** pour apprendre des interactions non linéaires ;
- **Autoencoders (VAE)** pour reconstruire les profils utilisateurs de manière latente ;
- **SASRec / Transformers séquentiels** pour capturer l'ordre temporel des films consommés.

Ces modèles répondraient en particulier aux limites identifiées sur la fusion linéaire.

6.3.2 Ajout de nouvelles sources de features

L'enrichissement des vecteurs items et utilisateurs pourrait bénéficier :

- d'embeddings Word2Vec ou BERT sur les descriptions complètes des films ;
- de signaux implicites (consultations, watch-time, clics, scroll) ;
- de features sociologiques ou démographiques additionnelles.

6.3.3 Réduction du biais de popularité

Plusieurs techniques sont adaptées :

- rééchantillonnage inversement proportionnel à la popularité ;
- régularisation ou pénalisation des items sur-recommandés ;
- diversification explicite via MMR (Maximal Marginal Relevance).

6.3.4 Optimisation pour un environnement de production

En vue d'un déploiement réel :

- remplacer la recherche de similarité brute par FAISS ou Annoy ;
- mettre en cache les top-K voisins items (matrice item-item) ;
- découpler entraînement (batch) et service en ligne ;
- monitorer la dérive et rafraîchir périodiquement les profils.

6.3.5 Évolution vers un système multi-étapes

Comme dans les systèmes industriels (Netflix, YouTube), on pourrait structurer :

1. un module **candidate generation** (rapide, massif) basé sur item-item enrichi ;
2. un module **ranking** (plus complexe) basé sur :
 - des features contextuelles,
 - des modèles neuronaux,
 - des signaux implicites.

Ce découpage améliorerait la pertinence globale et la scalabilité.

Conclusion du chapitre

L'approche hybride développée dans ce projet constitue un compromis efficace entre performance, interprétabilité et simplicité. Les résultats montrent une nette amélioration par rapport aux modèles individuels, mais laissent entrevoir plusieurs axes d'amélioration — notamment l'intégration de modèles neuronaux, la modélisation séquentielle et la réduction du biais de popularité — pour aller vers un système de recommandation proche des standards industriels.

Conclusion

Ce projet a permis de concevoir, implémenter et analyser un système de recommandation hybride appliqué au jeu de données MovieLens 1M. L’objectif était de dépasser les limites des approches collaboratives classiques en enrichissant les représentations items avec des informations de contenu, tout en intégrant un second signal basé sur la similarité entre utilisateurs. La méthodologie suivie — prétraitement rigoureux, modélisation modulaire, expérimentation contrôlée et analyse quantitative — a permis d’aboutir à un système robuste, cohérent et performant.

Le premier apport majeur de ce travail est la construction d’un modèle **item–item enrichi** combinant deux sources d’information complémentaires : (i) les cooccurrences issues du filtrage collaboratif et (ii) les attributs textuels et catégoriels des films (TF–IDF, genres encodés). Cette hybridation a amélioré la qualité des similarités, notamment pour les films peu notés, tout en atténuant partiellement le biais de popularité. Les résultats expérimentaux ont confirmé un gain significatif en NDCG@10 et en taux de hits par rapport à un item–item standard, montrant l’intérêt d’intégrer du contenu même minimal (titres et genres).

Le second apport réside dans l’ajout d’un module **user–user** combiné linéairement au modèle enrichi selon un ratio 90/10. Malgré un poids volontairement faible dans la fusion, ce signal utilisateur s’est révélé complémentaire : il introduit davantage de diversité, met en avant des films moins fréquents dans les profils similaires, et améliore le taux de découverte. Cette fusion simple mais efficace a permis d’obtenir un système globalement plus stable et plus riche que les approches individuelles, sans augmentation significative du coût computationnel.

Sur le plan expérimental, les métriques obtenues (NDCG, HR) montrent que le système répond convenablement aux objectifs initiaux : produire des recommandations pertinentes, diversifiées et relativement adaptées aux profils utilisateurs, tout en restant interprétable et léger. Le pipeline mis en place est modulable, reproductible et suffisamment générique pour être étendu vers des systèmes de recommandation industriels.

Néanmoins, plusieurs limites persistent. Le biais de popularité, bien que réduit, reste présent. Les profils utilisateurs peu actifs demeurent difficiles à modéliser avec précision. Enfin, la fusion entre modèles reste linéaire et ne capture qu’une partie des interactions complexes entre signaux comportementaux et signal de contenu. Ces limites ouvrent des perspectives naturelles pour de futurs travaux.

Plusieurs axes d’amélioration ont été identifiés. L’intégration de **méthodes neuronales** (NCF, autoencoders variationnels, modèles séquentiels type SASRec) permettrait d’apprendre des représentations plus expressives. L’exploitation de **features supplémentaires**, notamment issues des descriptions complètes des films ou de signaux implicites (clics, temps de visionnage), augmenterait la granularité du modèle. Enfin, une architecture multi-étapes proche des pratiques industrielles (candidate generation + ranking avancé) constituerait une évolution logique vers un système scalable et optimisé pour un

environnement réel.

En conclusion, ce travail démontre qu’une approche hybride, même relativement simple, peut significativement améliorer la performance des systèmes de recommandation basés sur MovieLens 1M. Le modèle obtenu constitue une base solide, fiable et extensible. Il ouvre la voie à des développements ultérieurs plus ambitieux, combinant richesse des données, modélisation avancée et exigences de production.