



Département : Mathématiques & Informatique

Filière : IID2 | Module : Machine Learning

Système de recommandation de films

Filtrage Collaboratif Item-Item (KNN) + Application Streamlit

Projet Machine Learning

Encadré par :

Professeur A.Ghazdali

Réalisé par :

Agoulzi Mohamed
Mouad El Abbassi

Table des matières

1	Contexte et idée du projet	2
1.1	Objectif principal	2
1.2	Livrables attendus	2
2	Données et description des fichiers	2
2.1	movies.csv	2
2.2	ratings.csv	3
2.3	Nature des données : interactions implicites vs explicites	3
3	Fondamentaux ML : définitions et choix méthodologiques	3
3.1	Définition du Machine Learning	3
3.2	À quel type de ML appartient notre modèle?	3
3.3	Définition de KNN	3
3.4	Métrique utilisée : similarité cosinus	3
3.5	Filtrage collaboratif vs filtrage basé sur le contenu	3
4	Pipeline complet : de la donnée à l'application	4
4.1	Étape 1 : Chargement et normalisation	4
4.2	Étape 2 : Filtrage (qualité & robustesse)	4
4.3	Étape 3 : Construction de la matrice Film $\times$ Utilisateur (creuse)	4
4.4	Étape 4 : Entraînement du modèle KNN	4
4.5	Étape 5 : Moteur de recommandation	5
4.6	Étape 6 : Gestion des cas limites (robustesse)	5
4.7	Étape 7 : Fallback "Popular Movies" (Cold Start)	5
5	Technologies utilisées et rôle de chaque composant	5
5.1	Python	5
5.2	Pandas & NumPy	5
5.3	SciPy Sparse (COO/CSR)	5
5.4	Scikit-learn	5
5.5	DiffLib (Fuzzy Search)	6
5.6	Streamlit	6
6	Architecture logicielle et structure du projet	6
6.1	Structure recommandée des fichiers	6
6.2	Séparation logique (bonnes pratiques)	6
6.3	Performance et cache	6
7	Évaluation qualitative et critères de réussite	6
7.1	Limites connues	7
7.2	Interface De Streamlit	7
8	Améliorations possibles (roadmap)	7
9	Conclusion	7

Résumé

Dans ce projet, nous avons développé un système de recommandation de films basé sur le **filtrage collaboratif Item-Item** (film-à-film). L'objectif est de recommander à l'utilisateur une liste de films pertinents à partir d'un titre saisi, même en présence d'erreurs de frappe, grâce à un mécanisme de **recherche approximative (fuzzy search)**. Notre solution s'appuie sur deux sources de données standard : `movies.csv` et `ratings.csv`. Nous avons conçu un pipeline robuste et optimisé pour les grands volumes : filtrage de bruit, construction de matrice creuse, apprentissage non supervisé via **k plus proches voisins (KNN)** et déploiement au travers d'une application **Streamlit** ergonomique. Le système propose également un mode **fallback** (films populaires) pour gérer les cas d'échec de recherche ou de données insuffisantes.

Mots-clés : Recommender Systems, Filtrage Collaboratif, Item-Based, KNN, Cosine Similarity, Matrices Creuses (CSR), Streamlit, Data Engineering.

1 Contexte et idée du projet

Les systèmes de recommandation sont omniprésents dans les plateformes de streaming, d'e-commerce et de musique : ils aident l'utilisateur à découvrir du contenu en réduisant le temps de recherche et en augmentant la satisfaction. Notre projet s'inscrit dans cette logique : **recommander des films en fonction des préférences observées chez des utilisateurs similaires**, à partir des notes attribuées.

1.1 Objectif principal

Notre objectif est de construire une solution logicielle complète qui :

- charge et traite efficacement des données de films et de notes,
- produit des recommandations **Item-Item** cohérentes (film-à-film),
- gère la saisie utilisateur (titres incomplets, fautes),
- affiche un résultat professionnel (titres, genres, score de similarité),
- propose une alternative fiable via un top films populaires.

1.2 Livrables attendus

Nous livrons :

- un pipeline de traitement des données (préparation, filtrage, modélisation),
- un moteur de recommandation Item-Item (KNN + cosine),
- une application Streamlit avec interface utilisateur.

2 Données et description des fichiers

Nous utilisons deux fichiers CSV, courants dans les datasets de recommandation.

2.1 `movies.csv`

Contient les informations descriptives des films :

- **movieId** : identifiant unique du film,
- **title** : titre (incluant souvent l'année),
- **genres** : liste de genres séparés par |.

2.2 ratings.csv

Contient les interactions utilisateurs :

- **userId** : identifiant unique utilisateur,
- **movieId** : identifiant du film noté,
- **rating** : note (ex. 0.5 à 5.0),
- **timestamp** : date/heure (non utilisée dans le modèle de base).

2.3 Nature des données : interactions implicites vs explicites

Les notes (*ratings*) sont des **feedbacks explicites**. Elles sont particulièrement informatives car elles représentent un jugement direct de l'utilisateur. Cela renforce la pertinence du filtrage collaboratif.

3 Fondamentaux ML : définitions et choix méthodologiques

3.1 Définition du Machine Learning

Le **Machine Learning** est un ensemble de méthodes permettant à un système d'apprendre des structures ou des règles à partir de données, afin de produire des prédictions ou décisions sans être explicitement programmé pour chaque cas.

3.2 À quel type de ML appartient notre modèle ?

Notre approche relève de l'**apprentissage non supervisé** :

- Nous n'avons pas une variable cible (label) à prédire,
- Nous cherchons plutôt des **proximités** / **similarités** entre films,
- L'algorithme KNN est utilisé ici comme une méthode de **recherche de voisinage** dans un espace vectoriel.

3.3 Définition de KNN

Le **k-Nearest Neighbors (KNN)** est une méthode basée sur la notion de distance :

- pour un objet donné, on recherche ses **k** objets les plus proches selon une métrique,
- dans un recommender Item-Item, l'objet est un **film** représenté par son vecteur de notes.

3.4 Métrique utilisée : similarité cosinus

Nous utilisons la **distance cosinus**, adaptée aux matrices creuses :

- elle compare l'orientation de deux vecteurs plutôt que leur amplitude,
- elle est robuste quand les utilisateurs n'ont noté qu'une petite fraction des films.

3.5 Filtrage collaboratif vs filtrage basé sur le contenu

Filtrage basé sur le contenu (Content-Based) : recommande des films similaires selon leurs caractéristiques (genres, synopsis, acteurs, etc.).

Filtrage collaboratif (Collaborative Filtering) : recommande en exploitant les comportements (notes) des utilisateurs.

Pourquoi notre choix Item-Based ?

- Il exploite directement la richesse des **notes utilisateurs** (plus informatives que des tags),

- Il est **plus proche des systèmes industriels** (Netflix/Amazon utilisent des signaux comportementaux),
- Il est performant et stable : la similarité film-à-film est réutilisable pour de nombreux utilisateurs,
- Il reste compatible avec une interface simple : “je saisis un film → je reçois des films proches”.

4 Pipeline complet : de la donnée à l’application

Nous avons structuré le projet comme un pipeline d’entreprise (MVP → itération).

4.1 Étape 1 : Chargement et normalisation

Nous chargeons `movies.csv` et `ratings.csv` en contrôlant les types (*dtypes*) afin de :

- réduire l’empreinte mémoire (ex. `int32`, `float32`),
- accélérer les opérations de jointure et d’agrégation,
- garantir la cohérence des identifiants `userId/movieId`.

4.2 Étape 2 : Filtrage (qualité & robustesse)

Nous appliquons deux filtres essentiels :

- **Min ratings par film** : on conserve les films ayant au moins un certain nombre de notes,
- **Min ratings par utilisateur** : on conserve les utilisateurs suffisamment actifs.

Pourquoi c’est crucial ?

- Les films très peu notés génèrent du bruit : on ne peut pas estimer une similarité fiable.
- Les utilisateurs qui ont très peu noté n’apportent pas d’information utile pour comparer des films.
- Le filtrage stabilise le modèle et réduit fortement la taille du problème (performance et mémoire).

4.3 Étape 3 : Construction de la matrice Film $\times$ Utilisateur (creuse)

Nous représentons chaque film comme un vecteur de notes utilisateurs. La matrice résultante est très **creuse** (beaucoup de zéros) car chaque utilisateur ne note qu’une petite partie des films.

Choix Data Engineering : matrices creuses CSR Nous utilisons un format **CSR (Compressed Sparse Row)** :

- nous stockons uniquement les valeurs non-nulles (notes réellement présentes),
- nous évitons de construire une énorme matrice dense, coûteuse en RAM,
- nous accélérons les opérations de voisinage pour KNN.

4.4 Étape 4 : Entraînement du modèle KNN

Nous entraînons un modèle **NearestNeighbors** sur la matrice CSR :

- **objectif** : trouver les films les plus proches d’un film donné,
- **sortie** : une liste de voisins + un score de distance (converti en similarité).

4.5 Étape 5 : Moteur de recommandation

Le moteur suit la logique :

1. l'utilisateur saisit un titre (avec ou sans fautes),
2. nous retrouvons le titre le plus proche via **fuzzy search**,
3. nous récupérons l'index du film dans la matrice,
4. nous cherchons ses voisins KNN,
5. nous retournons **Top-N recommandations** avec un score de similarité.

4.6 Étape 6 : Gestion des cas limites (robustesse)

Nous gérons proprement les cas suivants :

- titre introuvable (aucune correspondance fuzzy suffisamment proche),
- film filtré (pas assez de notes pour être dans le modèle),
- recommandations vides (données insuffisantes).

4.7 Étape 7 : Fallback “Popular Movies” (Cold Start)

Le problème de **cold start** apparaît lorsque l'utilisateur n'a pas d'historique ou que l'on ne peut pas calculer une similarité fiable. Notre fallback propose alors une liste de films populaires en combinant :

- nombre de notes (*popularité*),
- note moyenne (*qualité*).

5 Technologies utilisées et rôle de chaque composant

Notre solution combine des outils Data, ML et déploiement.

5.1 Python

Langage principal du projet : large écosystème Data Science, rapidité de prototypage, compatibilité Streamlit.

5.2 Pandas & NumPy

- **Pandas** : chargement, jointures, filtrage, agrégations (*groupby*) et préparation de données.
- **NumPy** : gestion efficace des tableaux et types numériques optimisés.

5.3 SciPy Sparse (COO/CSR)

- **COO** : format pratique pour construire une matrice à partir de triplets (ligne, colonne, valeur).
- **CSR** : format optimisé pour le stockage et les requêtes de voisinage.

5.4 Scikit-learn

- **NearestNeighbors** : implémentation stable et standard de KNN.
- **Distance cosinus** : mesure de proximité adaptée aux vecteurs creux.

5.5 Difflib (Fuzzy Search)

Nous utilisons un mécanisme de correspondance approximative afin :

- d’accepter les fautes de frappe,
- de supporter des titres partiels,
- d’améliorer fortement l’expérience utilisateur.

5.6 Streamlit

Framework de déploiement léger pour transformer un modèle ML en application interactive :

- interface simple (champ de saisie, bouton, tableau),
- affichage de recommandations,
- réglages de filtrage (min ratings) via la barre latérale,
- cache applicatif pour éviter de reconstruire le modèle à chaque interaction.

6 Architecture logicielle et structure du projet

Nous avons conçu une architecture claire, modulaire et maintenable.

6.1 Structure recommandée des fichiers

- `app.py` : application Streamlit (UI + orchestration),
- `movies.csv`, `ratings.csv` : données,
- `README.md` : instructions d’exécution, dépendances, exemples,
- (optionnel) `requirements.txt` : versions des librairies.

6.2 Séparation logique (bonnes pratiques)

- **Data Layer** : chargement, filtrage, préparation,
- **Model Layer** : entraînement KNN et recherche de voisins,
- **Service Layer** : fonction de recommandation + fuzzy search,
- **UI Layer** : Streamlit (interaction utilisateur, affichage).

6.3 Performance et cache

Nous utilisons un mécanisme de cache pour :

- éviter de recalculer le modèle à chaque refresh Streamlit,
- améliorer l’expérience utilisateur,
- stabiliser les temps de réponse.

7 Évaluation qualitative et critères de réussite

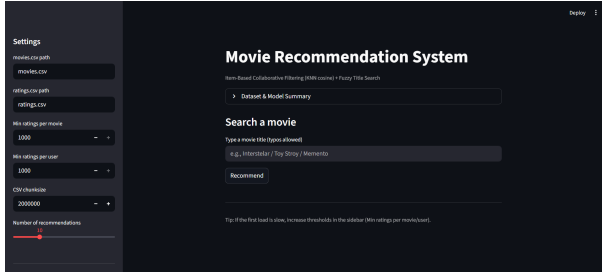
Nous évaluons le système selon des critères concrets :

- **Pertinence** : recommandations cohérentes avec le film recherché,
- **Robustesse** : bonne gestion des titres mal saisis,
- **Stabilité** : pas de crash mémoire grâce au format creux,
- **Explicabilité** : affichage des genres et score de similarité,
- **Utilisabilité** : interface claire et rapide.

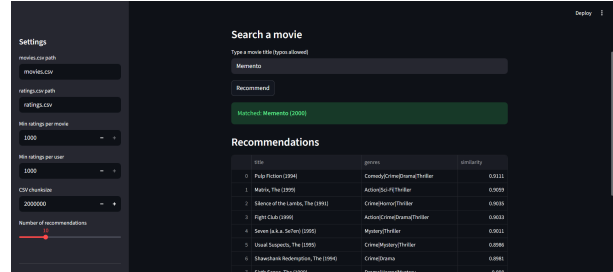
7.1 Limites connues

- **Biais de popularité** : les films très populaires peuvent apparaître souvent.
- **Cold start** : nouveaux films ou nouveaux utilisateurs peu notés.
- **Ambiguïté des titres** : certains titres peuvent exister en plusieurs versions.

7.2 Interface De Streamlit



(a) First Interface



(b) Test de Modele

FIGURE 1 – Interfaces de l'application

8 Améliorations possibles (roadmap)

Si nous disposons de plus de temps, nous pourrions :

- ajouter un mode **User-Based** ou une recommandation personnalisée (par `userId`),
- intégrer des métadonnées externes (synopsis, acteurs) pour un modèle hybride,
- réduire le biais de popularité via des pondérations (*shrinkage*, normalisation),
- persister le modèle (sauvegarde) pour un démarrage instantané de l'app,
- déployer sur Streamlit Cloud / Docker avec un pipeline reproductible.

9 Conclusion

Dans ce projet, nous avons conçu et implémenté un système de recommandation de films **professionnel** et **robuste**, basé sur le filtrage collaboratif Item-Item. Notre pipeline met l'accent sur les bonnes pratiques Data Engineering (gestion mémoire, matrices creuses, filtrage qualité) et sur l'expérience utilisateur (fuzzy search, interface Streamlit, fallback popularité). Le résultat est une application exploitable, claire, et alignée avec les attentes d'un projet académique orienté Data Engineering et Machine Learning.