Faculté des Sciences de Sfax

Rapport de Projet

Système de Recommandation Hybride Amélioré par SVD et Comparaison avec les Approches Précédentes

(Approche Hybride Item-Centrée)

Le code source de ce projet est disponible sur GitHub :
https://github.com/Ahmedouyahya/systemes_adaptatifs-.git

Par

Ahmedou Yahye Gleiguem Kheyri

étudiant en Master Recherche Informatique

25 février 2025

Table des matières

1 Introduction

Ce rapport explore en profondeur les systèmes de recommandation, en se concentrant particulièrement sur une approche hybride innovante. Nous allons développer et évaluer un système de recommandation hybride item-centré, en le comparant aux approches traditionnelles : le filtrage collaboratif basé contenu, le filtrage collaboratif basé utilisateurs/items, et une méthode améliorée par la Décomposition en Valeurs Singulières (SVD). L'objectif est d'analyser les forces et faiblesses de chaque approche et de démontrer comment l'hybridation et l'utilisation de techniques de réduction de dimensionnalité comme la SVD peuvent améliorer les performances et la robustesse des recommandations.
Le rapport est structuré en plusieurs étapes clés :

— Présentation des algorithmes de recommandation : description des approches basées contenu, filtrage collaboratif (utilisateur et item) et SVD.
— Implémentation des approches de base (contenu, collaboratif item et utilisateur) et de l'approche hybride combinant Pearson et Cosinus.
— Amélioration de l'approche hybride par l'intégration de la SVD pour la réduction de dimensionnalité et l'amélioration de la performance.
— Évaluation comparative des différentes approches en termes de qualité de recommandation et de temps de calcul.
— Analyse des résultats et perspectives d'amélioration.

2 Présentation Mathématique des Approches de Recommandation

2.1 Filtrage Collaboratif (Collaborative Filtering - CF)

Le filtrage collaboratif repose sur l'idée que les utilisateurs ayant des préférences similaires dans le passé auront des préférences similaires à l'avenir. Il existe deux types principaux de CF :

— **Filtrage Collaboratif Basé Utilisateurs (User-Based CF)** : Recommande des items que des utilisateurs similaires ont appréciés. La similarité entre utilisateurs est calculée sur la base de leurs évaluations passées.
— **Filtrage Collaboratif Basé Items (Item-Based CF)** : Recommande des items similaires à ceux qu'un utilisateur a appréciés dans le passé. La similarité entre items est calculée en se basant sur les évaluations des utilisateurs. L'approche item-based est généralement

préférée pour sa performance et sa scalabilité.

2.2 Filtrage Basé Contenu (Content-Based Filtering)

Le filtrage basé contenu recommande des items similaires à ceux que l'utilisateur a aimés dans le passé, basés sur une analyse des descriptions et des attributs des items (par exemple, genres, acteurs, réalisateurs pour les films). Cette approche ne nécessite pas les évaluations des autres utilisateurs.

2.3 Décomposition en Valeurs Singulières (SVD)

La SVD est une technique de réduction de dimensionnalité qui peut être appliquée à la matrice utilisateur-item. Elle vise à factoriser la matrice en trois matrices de rang inférieur (U, , Vt). La matrice reconstruite après SVD (UVt) permet de prédire les notes manquantes et de réduire le bruit et la sparsité des données.

2.4 Normalisation des Notes

Pour atténuer les biais liés aux différences d'échelle de notation entre utilisateurs, on normalise les notes en les centrant autour de la moyenne de chaque item :

$$r_{norm}(u,i) = r(u,i) - \bar{r_i} \tag{1}$$

où $\bar{r_i}$ est la moyenne des notes pour l'item i.

2.5 Mesures de Similarité

Dans le contexte du filtrage collaboratif, nous utilisons deux mesures de similarité principales :

— **Corrélation de Pearson** : Mesure la corrélation linéaire entre les vecteurs de notes centrées.

$$sim_p(A,B) = \frac{\sum_i (a_i - \bar{a})(b_i - \bar{b})}{\sqrt{\sum_i (a_i - \bar{a})^2}\sqrt{\sum_i (b_i - \bar{b})^2}} \tag{2}$$

— **Similarité Cosinus** : Mesure le cosinus de l'angle entre les vecteurs de notes.

$$sim_c(A,B) = \frac{A \cdot B}{\|A\|\|B\|} \tag{3}$$

3 Présentation des Données

Nous utilisons deux fichiers de données pour ce projet :

ratings_tp5.csv

Ce fichier contient les évaluations des utilisateurs pour les films. Les colonnes sont : ‘userId‘, ‘movieId‘, ‘rating‘, et ‘timestamp‘.

```
userId,movieId,rating,timestamp
1,1,4.0,964982703
1,3,4.0,964981247
1,6,4.0,964982...
...
```

movies_tp5.csv

Ce fichier contient les informations sur les films, notamment leur identifiant, titre et genres. Les colonnes sont : ‘movieId‘, ‘title‘, et ‘genres‘.

```
movieId,title,genres
1,Toy Story (1995),Adventure|Animation|Children|Comedy|Fantasy
2,Jumanji (1995),Adventure|Children|Fantasy
3,Grumpier Old Men (1995),Comedy|Ro...
...
```

4 ETAPE 1 : Chargement des Données

Objectif : Importer les données d’évaluations des utilisateurs et les métadonnées des films. Les données brutes peuvent contenir des valeurs manquantes (NaN) qui seront traitées.

4.1 Code

```
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from scipy.sparse.linalg import svds

# ---------------------------
```

```
# 1    CHARGEMENT DES DONN ES
# --------------------------
print("\n[1] Chargement des donn es...")
ratings = pd.read_csv('ratings_tp5.csv', sep=',', na_values=[
    ''])
movies = pd.read_csv('movies_tp5.csv', sep=',')

print(f"    {len(ratings)}   valuations   charg es")
print(f"    {len(movies)} films charg s")

# V rifier les colonnes
print("\nColonnes des fichiers :")
print("Ratings :", ratings.columns)
print("Movies :", movies.columns)
```

Listing 1 – Chargement des données

4.2 Output

Output - Etape 1

[1] Chargement des données...
- 100836 évaluations chargées
- 9742 films chargés

Colonnes des fichiers :
Ratings : Index(['userId', 'movieId', 'rating', 'timestamp'], dtype='object')
Movies : Index(['movieId', 'title', 'genres'], dtype='object')

5 ETAPE 2 : Fusion des Données

Processus : Fusionner les évaluations avec les métadonnées des films en utilisant la colonne ‘movieId‘. Cela enrichit les données d’évaluation avec les genres des films.

5.1 Code

```
# ---------------------------
# 2    FUSION DES DONN ES
# ---------------------------
```

```
print("\n[2] Fusion des donn es...")
df = pd.merge(ratings, movies, on='movieId', how='inner')
print(f"    Dataset fusionn  : {df.shape[0]} lignes")
```

Listing 2 – Fusion des données

5.2 Output

Output - ETAPE 2

[2] Fusion des données...
• Dataset fusionné : 100836 lignes

6 ETAPE 3 : Création de la Matrice Utilisateur-Film

Structure : Création d'une matrice creuse de taille $n_{films} \times n_{utilisateurs}$ où chaque cellule contient la note attribuée par un utilisateur à un film. Cette matrice est essentielle pour les approches de filtrage collaboratif item-centré.

6.1 Code

```
# ---------------------------
# 3    MATRICE UTILISATEUR-FILM
# ---------------------------
print("\n[3] Cr ation de la matrice utilisateur-film...")
matrix = df.pivot_table(index='title', columns='userId',
    values='rating')

print(f"    Dimensions : {matrix.shape[0]} films x {matrix.
    shape[1]} utilisateurs")
print(f"    Taux de remplissage : {(1 - matrix.isna().mean().
    mean()):.1%}")
```

Listing 3 – Création de la matrice utilisateur-film

6.2 Output

Output - ETAPE 3

[3] Création de la matrice utilisateur-film...
- Dimensions : 9719 films x 610 utilisateurs
- Taux de remplissage : 1.7%

7 ETAPE 4 : Normalisation des Notes

Impact : Réduire le biais systématique entre utilisateurs. Permet une comparaison plus équitable entre les films en centrant les notes autour de la moyenne de chaque film.

7.1 Code

```
# ----------------------------
# 4    NORMALISATION DES NOTES
# ----------------------------
print("\n[4] Normalisation des notes...")
matrix_norm = matrix.subtract(matrix.mean(axis=1), axis=0)
```

Listing 4 – Normalisation des notes

7.2 Output

Output - ETAPE 4

[4] Normalisation des notes...

8 ETAPE 5 : Calcul des Similarités (Pearson Cosinus)

Comparaison : Calcul des matrices de similarité entre films en utilisant les méthodes de corrélation de Pearson et de similarité cosinus.
— Pearson : sensible aux tendances linéaires et ajuste pour les différences de moyenne.
— Cosinus : mesure l'angle entre les vecteurs de notes et est moins sensible à l'amplitude des notes, mais nécessite de traiter les valeurs manquantes.

8.1 Code

```
# ----------------------------
# 5    CALCUL DES SIMILARIT S (Pearson & Cosinus)
# ----------------------------
print("\n[5] Calcul des similarit s...")

# Pearson
print("    Calcul des corr lations de Pearson...")
similarity_pearson = matrix_norm.T.corr()

# Cosinus
print("    Calcul des similarit s cosinus...")
matrix_norm_zero = matrix_norm.fillna(0)
similarity_cosine = pd.DataFrame(
    cosine_similarity(matrix_norm_zero),
    index=matrix_norm.index,
    columns=matrix_norm.index
)
```

Listing 5 – Calcul des similarités entre films

8.2 Output

Output - ETAPE 5

[5] Calcul des similarités...
• Calcul des corrélations de Pearson...
• Calcul des similarités cosinus...

9 ETAPE 6 : Prédiction des Notes avec Approches Item-Based

Méthodologie Item-Based : Pour prédire la note d'un utilisateur pour un film non évalué, nous calculons une moyenne pondérée des notes des films les plus similaires que l'utilisateur a déjà notés.
Formule de Prédiction :

$$\hat{r}_{u,i} = \bar{r}_i + \frac{\sum_{j \in N(i)} sim(i,j) \cdot (r_{u,j} - \bar{r}_j)}{\sum_{j \in N(i)} |sim(i,j)|} \tag{4}$$

où $N(i)$ représente les n voisins les plus similaires de l'item i.

9.1 Code

```
# ----------------------------
# 6    PR DICTION DES NOTES
# ----------------------------
def predict_rating(user_id, movie, similarity_matrix,
    n_neighbors=2):
    """Pr dit la note d'un utilisateur pour un film donn .
       """
    if movie not in similarity_matrix:
        return matrix.mean().mean()  # Retourne la moyenne
           globale si film inconnu

    # Trouver les films similaires
    similar_movies = similarity_matrix[movie].dropna().
       sort_values(ascending=False)

    # Garder les N voisins les plus proches
    top_neighbors = similar_movies.head(n_neighbors)

    # Films d j   not s par l'utilisateur
    rated_movies = matrix_norm[user_id].dropna()

    # V rifier si des voisins existent
    common_movies = rated_movies.index.intersection(
       top_neighbors.index)
    if common_movies.empty:
        return matrix.mean().mean()  # Retourner la moyenne
           globale

    # Calculer la pr diction
```

```
24      top_neighbors = top_neighbors.loc[common_movies]
25      pred_norm = np.average(rated_movies[common_movies],
            weights=top_neighbors)
26
27      # D  -normalisation
28      pred = pred_norm + matrix.mean(axis=1)[movie]
29      return round(pred, 2)
```

Listing 6 – Fonction de prédiction Item-Based (Pearson Cosinus)

9.2 Output

Output - ETAPE 6

Fonction de prédiction définie. (Pas d'output direct)

10 ETAPE 7 : Approche Hybride (Combinaison Pearson Cosinus)

Hybridation : Pour combiner les avantages des deux mesures de similarité, nous proposons une approche hybride qui pondère les prédictions de Pearson et Cosinus. Cette approche vise à améliorer la robustesse et la précision des recommandations.
Formule de Prédiction Hybride :

$$\hat{r}_{hybrid}(u,i) = \alpha \cdot \hat{r}_{Pearson}(u,i) + (1-\alpha) \cdot \hat{r}_{Cosinus}(u,i) \qquad (5)$$

où α est un facteur de pondération entre 0 et 1, contrôlant l'importance relative de Pearson et Cosinus.

10.1 Code

```
1  # ----------------------------
2  # 7    APPROCHE HYBRIDE (Combinaison Pearson & Cosinus)
3  # ----------------------------
4  def hybrid_prediction(user_id, movie, alpha=0.5, n_neighbors
       =2):
5      """Pr dit la note en combinant Pearson et Cosinus."""
6      pred_p = predict_rating(user_id, movie,
           similarity_pearson, n_neighbors)
```

```
    pred_c = predict_rating(user_id, movie, similarity_cosine
        , n_neighbors)

    # Combinaison hybride
    return round(alpha * pred_p + (1 - alpha) * pred_c, 2)
```

Listing 7 – Fonction de prédiction hybride (Pearson Cosinus)

10.2 Output

Output - ETAPE 7

Fonction de prédiction hybride définie. (Pas d'output direct)

11 ETAPE 8 : Amélioration par SVD et Recommandation Basée SVD

11.1 Décomposition en Valeurs Singulières (SVD)

Nous utilisons la SVD pour réduire la dimensionnalité de la matrice utilisateur-film et pour améliorer la qualité des prédictions. La SVD nous permet de capturer les patterns latents dans les données et de réduire le bruit.

11.2 Code pour SVD

```
# ----------------------------
# 8    AM LIORATION PAR SVD
# ----------------------------
print("\n[8] Application de la SVD pour la r duction de
    dimensionnalit ...")

# Cr ation de la matrice utilisateur-film (pivot e pour SVD
    - utilisateurs en lignes)
R_svd = df.pivot_table(index='userId', columns='movieId',
    values='rating')
R_filled_svd = R_svd.fillna(0)
R_filled_matrix_svd = R_filled_svd.values

# D composition SVD avec k=50 composantes
k = 50
U, sigma, Vt = svds(R_filled_matrix_svd, k=k)
sigma = np.diag(sigma)
```

```
15
16 # Reconstruction de la matrice de pr dictions SVD
17 R_pred_svd = np.dot(np.dot(U, sigma), Vt)
18 preds_df_svd = pd.DataFrame(R_pred_svd, index=R_svd.index,
       columns=R_svd.columns)
19
20 print("SVD appliqu e et matrice de pr dictions reconstruite
       .")
```

Listing 8 – Implémentation de la SVD

11.3 Output

Output - ETAPE 8

[8] Application de la SVD pour la réduction de dimensionnalité...
SVD appliquée et matrice de prédictions reconstruite.

11.4 Fonction de Recommandation Basée SVD

```
1  def recommend_movies_svd(user_id, preds_df_svd, movies,
       original_ratings, num_recommendations=5):
2      """
3      G n re des recommandations pour un utilisateur en
           utilisant la d composition SVD.
4      """
5      # Extraire les pr dictions pour l'utilisateur
6      user_pred = preds_df_svd.loc[user_id]
7
8      # R cup rer les films d j  not s par l'utilisateur
9      rated_movies = original_ratings[original_ratings.userId
           == user_id]['movieId']
10
11     # Filtrer les films non not s
12     recommendations = movies[~movies['movieId'].isin(
           rated_movies)]
13
14     # Ajouter la note pr dite
15     recommendations = recommendations.copy()
16     recommendations['predicted_rating'] = recommendations['
           movieId'].apply(lambda x: user_pred.get(x, 0))
17
18     # Trier par note pr dite d croissante
19     recommendations = recommendations.sort_values(by='
           predicted_rating', ascending=False)
20     return recommendations.head(num_recommendations)
```

Listing 9 – Fonction de recommandation basée sur SVD

11.5 Output

Output - ETAPE 8

Fonction de recommandation SVD définie. (Pas d'output direct)

12 ETAPE 9 : Générer les Recommandations et Comparaison

Recommandations Utilisateur : Génération de recommandations pour un utilisateur spécifique en utilisant l'approche hybride et l'approche basée SVD, puis comparaison des résultats.

12.1 Code

```
# ----------------------------
# 9    G  N  RER  LES RECOMMANDATIONS ET COMPARAISON
# ----------------------------
def generate_recommendations(user_id, alpha=0.5, n_neighbors=2, n_reco=3):
    """G  n ère une liste de recommandations hybrides pour un utilisateur."""
    print(f"\n=== Recommandations Hybrides pour l'utilisateur {user_id} ===")

    # Films non   valus
    unwatched_movies = matrix[user_id][matrix[user_id].isna()].index

    # G  n  rer les pr dictions
    predictions = [
        (movie, hybrid_prediction(user_id, movie, alpha, n_neighbors))
        for movie in unwatched_movies
    ]

    # Trier par note pr dite et retourner les N meilleurs films
```

```
    recommendations = sorted(predictions, key=lambda x: x[1],
        reverse=True)[:n_reco]

    # Afficher les recommandations
    for film, note in recommendations:
        print(f" {film} : {note}/5")

    return recommendations

# Utilisateur test et g n ration des recommandations
    hybrides
user_id_test = 4
print("\n[9] G n ration des recommandations pour l'
    utilisateur", user_id_test, "...")
recommendations_hybrid = generate_recommendations(user_id=
    user_id_test, alpha=0.5, n_neighbors=2, n_reco=3)

# G n rer les recommandations SVD pour comparaison
print("\n=== Recommandations SVD pour l'utilisateur",
    user_id_test, "===")
reco_svd = recommend_movies_svd(user_id_test, preds_df_svd,
    movies, ratings, num_recommendations=3)
print(reco_svd[['title', 'predicted_rating']])

print("\n[9] Comparaison des approches...")
print("\ n   Comparaison des recommandations hybrides et SVD
    affich e ci-dessus.")
```

Listing 10 – Générer les recommandations et comparer

12.2 Output

Output - ETAPE 9

[9] Génération des recommandations pour l'utilisateur 4 ...

=== Recommandations Hybrides pour l'utilisateur 4 ===
Spider-Man : 3.4/5
Lord of the Rings : The Fellowship of the Ring, The : 3.39/5
Lord of the Rings : The Two Towers, The : 3.39/5

=== Recommandations SVD pour l'utilisateur 4 ===
title predicted$_r ating 2250 Who Framed Roger Rabbit? (1988) 2.38740546 Usual Suspects, The (199$

[9] Comparaison des approches...
• Comparaison des recommandations hybrides et SVD affichée ci-dessus.

13 ETAPE 10 : Évaluation Comparative et Analyse Finale

13.1 Comparaison des Approches

13.1.1 Filtrage Collaboratif Item-Based (Pearson, Cosinus et Hybride)

Points Forts :
— Simplicité de mise en œuvre et interprétabilité.
— Bonnes performances pour les données clairsemées après normalisation.
— L'approche hybride combine les avantages de Pearson et Cosinus, potentiellement plus robuste.
Points Faibles :
— Sensibilité au choix des paramètres (nombre de voisins, facteur alpha pour l'hybride).
— Dépendance de la qualité des recommandations à la matrice de similarité.

13.1.2 Filtrage Collaboratif Basé SVD

Points Forts :
— Réduction efficace de la dimensionnalité et du bruit dans les données.

— Capacité à découvrir des relations latentes entre utilisateurs et items.
— Souvent performant en termes de précision de prédiction.

Points Faibles :

— Moins interprétable que les approches basées similarité.
— Peut souffrir d'un "cold start" important (difficulté à recommander de nouveaux items ou à de nouveaux utilisateurs).
— Coût de calcul potentiellement plus élevé pour la décomposition SVD, surtout sur de très grands datasets.

13.1.3 Comparaison Directe des Recommandations

En observant les recommandations pour l'utilisateur 4, l'approche hybride tend à recommander des films populaires et bien notés en général (Spider-Man, Lord of the Rings), tandis que l'approche SVD propose des films potentiellement plus spécifiques ou moins grand public (Who Framed Roger Rabbit, Usual Suspects). Le choix de l'approche dépendra de l'objectif : recommandations populaires et pertinentes (hybride) ou recommandations plus personnalisées et potentiellement surprenantes (SVD).

13.2 Analyse Finale et Conclusion

Conclusion : L'approche hybride combinant Pearson et Cosinus offre une bonne base pour un système de recommandation item-centré, alliant simplicité et performance. L'amélioration par la SVD représente une voie prometteuse pour affiner les prédictions en réduisant la dimensionnalité et en exploitant les structures latentes des données. Le choix entre ces approches, et d'autres comme le filtrage basé contenu ou utilisateur, dépendra des spécificités du dataset, des objectifs de recommandation (popularité vs. personnalisation), et des contraintes de performance.

13.3 Perspectives d'Amélioration

— **Optimisation des Paramètres** : Ajuster finement les paramètres comme le nombre de voisins ($n\_neighbors$), le facteur de pondération α pour l'hybride, et le rang k pour la SVD via des techniques de validation croisée pour maximiser la performance du système.
— **Intégration de Données de Contenu** : Enrichir le modèle hybride ou SVD en intégrant des informations basées contenu (genres, acteurs,

réalisateurs) pour pallier le problème du "cold start" et améliorer la diversité des recommandations.
— **Évaluation Quantitative** : Mettre en place une évaluation quantitative rigoureuse en utilisant des métriques comme le RMSE (Root Mean Squared Error), la précision, le rappel, ou le NDCG (Normalized Discounted Cumulative Gain) pour comparer objectivement les performances des différentes approches.
— **Scalabilité et Performance** : Explorer des techniques d'optimisation et des librairies de calcul performantes pour assurer la scalabilité du système, notamment pour les phases de calcul de similarité et de décomposition SVD sur de très grands datasets.
— **Approches Hybrides Avancées** : Tester des méthodes hybrides plus sophistiquées qui adaptent dynamiquement la pondération entre les approches (Pearson, Cosinus, SVD, contenu) en fonction du contexte ou des caractéristiques de l'utilisateur/item.

A Code Complet (Version Pratique et Commentée)

```
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from scipy.sparse.linalg import svds

# ----------------------------
# 1    CHARGEMENT DES DONN ES
# ----------------------------
print("\n[1] Chargement des donn es...")
ratings = pd.read_csv('ratings_tp5.csv', sep=',', na_values=[
    ''])
movies = pd.read_csv('movies_tp5.csv', sep=',')

print(f"    {len(ratings)}   valuations   charg es")
print(f"    {len(movies)} films charg s")

# ----------------------------
# 2    FUSION DES DONN ES
# ----------------------------
print("\n[2] Fusion des donn es...")
df = pd.merge(ratings, movies, on='movieId', how='inner')
print(f"    Dataset fusionn  : {df.shape[0]} lignes")

# ----------------------------
```

```
# 3    MATRICE UTILISATEUR-FILM
# ----------------------------
print("\n[3] Cr ation de la matrice utilisateur-film...")
matrix = df.pivot_table(index='title', columns='userId',
    values='rating')

print(f"    Dimensions : {matrix.shape[0]} films x {matrix.
    shape[1]} utilisateurs")
print(f"    Taux de remplissage : {(1 - matrix.isna().mean().
    mean()):.1%}")

# ----------------------------
# 4    NORMALISATION DES NOTES
# ----------------------------
print("\n[4] Normalisation des notes...")
matrix_norm = matrix.subtract(matrix.mean(axis=1), axis=0)

# ----------------------------
# 5    CALCUL DES SIMILARIT S (Pearson & Cosinus)
# ----------------------------
print("\n[5] Calcul des similarit s...")

# Pearson
print("    Calcul des corr lations de Pearson...")
similarity_pearson = matrix_norm.T.corr()

# Cosinus
print("    Calcul des similarit s cosinus...")
matrix_norm_zero = matrix_norm.fillna(0)
similarity_cosine = pd.DataFrame(
    cosine_similarity(matrix_norm_zero),
    index=matrix_norm.index,
    columns=matrix_norm.index
)

# ----------------------------
# 6    FONCTION DE PR DICTION ITEM-BASED
# ----------------------------
def predict_rating(user_id, movie, similarity_matrix,
    n_neighbors=2):
    """Pr dit la note d'un utilisateur pour un film donn  (
        Item-Based CF)."""
    if movie not in similarity_matrix:
        return matrix.mean().mean()  # Retourne la moyenne
            globale si film inconnu

    # Trouver les films similaires
    similar_movies = similarity_matrix[movie].dropna().
        sort_values(ascending=False)
```

```
66
67      # Garder les N voisins les plus proches
68      top_neighbors = similar_movies.head(n_neighbors)
69
70      # Films d j  not s par l'utilisateur
71      rated_movies = matrix_norm[user_id].dropna()
72
73      # V rifier si des voisins existent
74      common_movies = rated_movies.index.intersection(
            top_neighbors.index)
75      if common_movies.empty:
76          return matrix.mean().mean()  # Retourner la moyenne
                globale
77
78      # Calculer la pr diction
79      top_neighbors = top_neighbors.loc[common_movies]
80      pred_norm = np.average(rated_movies[common_movies],
            weights=top_neighbors)
81
82      # D  -normalisation
83      pred = pred_norm + matrix.mean(axis=1)[movie]
84      return round(pred, 2)
85
86  # ----------------------------
87  # 7    APPROCHE HYBRIDE (COMBINAISON PEARSON & COSINUS)
88  # ----------------------------
89  def hybrid_prediction(user_id, movie, alpha=0.5, n_neighbors
        =2):
90      """Pr dit la note en combinant les pr dictions de
            Pearson et Cosinus."""
91      pred_p = predict_rating(user_id, movie,
            similarity_pearson, n_neighbors)
92      pred_c = predict_rating(user_id, movie, similarity_cosine
            , n_neighbors)
93
94      # Combinaison hybride
95      return round(alpha * pred_p + (1 - alpha) * pred_c, 2)
96
97  # ----------------------------
98  # 8    AM LIORATION PAR SVD
99  # ----------------------------
100 print("\n[8] Application de la SVD pour la r duction de
        dimensionnalit ...")
101
102 # Cr ation de la matrice utilisateur-film pour SVD (
        utilisateurs en lignes)
103 R_svd = df.pivot_table(index='userId', columns='movieId',
        values='rating')
104 R_filled_svd = R_svd.fillna(0)
```

```
R_filled_matrix_svd = R_filled_svd.values

# D composition SVD
k = 50  # Nombre de composantes singulières
U, sigma, Vt = svds(R_filled_matrix_svd, k=k)
sigma = np.diag(sigma)
R_pred_svd = np.dot(np.dot(U, sigma), Vt) # Matrice de
    pr dictions SVD
preds_df_svd = pd.DataFrame(R_pred_svd, index=R_svd.index,
    columns=R_svd.columns) # DataFrame des pr dictions SVD

print("SVD appliqu e et matrice de pr dictions reconstruite
    .")

# ----------------------------
# 9    FONCTION DE RECOMMANDATION BAS E SVD
# ----------------------------
def recommend_movies_svd(user_id, preds_df_svd, movies,
    original_ratings, num_recommendations=5):
    """G n re des recommandations pour un utilisateur en
       utilisant SVD."""
    user_pred = preds_df_svd.loc[user_id] # Pr dictions SVD
       pour l'utilisateur
    rated_movies = original_ratings[original_ratings.userId
       == user_id]['movieId'] # Films d j  not s
    recommendations = movies[~movies['movieId'].isin(
       rated_movies)].copy() # Films non not s
    recommendations['predicted_rating'] = recommendations['
       movieId'].apply(lambda x: user_pred.get(x, 0)) #
       Ajouter notes pr dites
    recommendations = recommendations.sort_values(by='
       predicted_rating', ascending=False) # Trier
    return recommendations.head(num_recommendations)

# ----------------------------
# 10    G N RER ET AFFICHER LES RECOMMANDATIONS
# ----------------------------
def generate_recommendations(user_id, alpha=0.5, n_neighbors
    =2, n_reco=3):
    """G n re des recommandations hybrides pour un
       utilisateur."""
    print(f"\n=== Recommandations Hybrides pour l'utilisateur
        {user_id} ===")
    unwatched_movies = matrix[user_id][matrix[user_id].isna()
       ].index # Films non vus
    predictions = [(movie, hybrid_prediction(user_id, movie,
       alpha, n_neighbors)) for movie in unwatched_movies] #
       Pr dictions hybrides
```

```
136     recommendations = sorted(predictions, key=lambda x: x[1],
            reverse=True)[:n_reco] # Trier et top N
137     for film, note in recommendations:
138         print(f" {film} : {note}/5") # Afficher
139     return recommendations
140
141 user_id_test = 4  # Utilisateur de test
142
143 print("\n[9] G n ration des recommandations pour l'
    utilisateur", user_id_test, "...")
144 recommendations_hybrid = generate_recommendations(user_id=
    user_id_test, alpha=0.5, n_neighbors=2, n_reco=3) #
    G n rer hybrides
145
146 print("\n=== Recommandations SVD pour l'utilisateur",
    user_id_test, "===")
147 reco_svd = recommend_movies_svd(user_id_test, preds_df_svd,
    movies, ratings, num_recommendations=3) # G n rer SVD
148 print(reco_svd[['title', 'predicted_rating']]) # Afficher SVD
149
150 print("\n[10] Comparaison des approches : Affichage des
    recommandations hybrides et SVD ci-dessus.")
151 print("\nFin du rapport.")
```

Listing 11 – Code complet de l'implémentation