

Assignment No 3

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.decomposition import NMF

movies = pd.read_csv("/content/RS-A2_A3_movie.csv")
tags = pd.read_csv("/content/RS-A2_A3_tag.csv")
ratings = pd.read_csv("/content/RS-A2_A3_Filtered_Ratings.csv")

merged = pd.merge(movies, tags[['movieId', 'tag']], on='movieId', how='left').fillna({'tag': ''})
merged['content'] = merged['genres'] + " " + merged['tag']

tfidf = TfidfVectorizer(stop_words='english')
content_sim = cosine_similarity(tfidf.fit_transform(merged['content']))
content_df = pd.DataFrame(content_sim, index=merged['movieId'],
                           columns=merged['movieId'])

pivot = ratings.pivot_table(index='userId', columns='movieId', values='rating').fillna(0)
nmf = NMF(n_components=10, random_state=42)
movie_factors = nmf.fit_transform(pivot.T)
collab_df = pd.DataFrame(cosine_similarity(movie_factors),
                          index=pivot.columns, columns=pivot.columns)

common = content_df.index.intersection(collab_df.index)
content_df = content_df.loc[common, common]
collab_df = collab_df.loc[common, common]

hybrid = 0.6 * content_df + 0.4 * collab_df

def recommend(title, n=5):
    m_id = merged.loc[merged['title'] == title, 'movieId'].values[0]
    if m_id not in hybrid.index:
        print("Movie not found in both datasets.")
        return
    scores = hybrid[m_id].sort_values(ascending=False)[1:n+1]
    print(f"\nTop {n} recommendations for '{title}':")
    for mid in scores.index:
        print("-", merged.loc[merged['movieId'] == mid, 'title'].values[0])

recommend("Toy Story (1995)")
```

Viva-Ready Questions & Answers

Q: What is a hybrid recommendation system?

A: It combines content-based filtering (movie attributes) and collaborative filtering (user ratings) to improve accuracy and overcome the limitations of both individual methods.

Q: Why use a hybrid approach?

A: It handles cold-start problems, increases diversity, and gives more balanced recommendations by blending user preferences and movie content.

Q: Which two techniques are combined in this code?

A:

- 1. TF-IDF + Cosine Similarity (Content-based)**
 - 2. NMF (Non-negative Matrix Factorization) (Collaborative filtering)**
-

Q: What is the role of TF-IDF in this system?

A: It transforms movie genres and tags into numerical vectors to measure content similarity between movies.

Q: What is NMF, and why is it used here?

A: NMF (Non-negative Matrix Factorization) is a matrix decomposition technique used to find hidden user–item factors in the rating matrix for collaborative filtering.

Q: What is the advantage of using cosine similarity again after NMF?

A: After factorization, cosine similarity helps measure the closeness of movies in the latent feature space, identifying which movies are similar in users' preferences.

Q: What are the blending weights (0.6 and 0.4) for?

A: They determine the contribution of each system — here, 60% weight to content-based similarity and 40% to collaborative filtering.

Q: What is the meaning of the pivot table step?

A: The pivot table converts user ratings into a matrix format where rows = users and columns = movies, required for NMF to find latent factors.

Q: What are latent factors in collaborative filtering?

A: Latent factors represent hidden features learned from user–item interactions, like movie popularity, genre preference, or rating behavior.

Q: What happens in the alignment step (common intersection)?

A: It ensures both content and collaborative similarity matrices use the same movie IDs, allowing proper blending.

Q: How is the final hybrid similarity computed?

A: By taking a weighted average of both similarity matrices:

Hybrid=0.6×Content Similarity+0.4×Collaborative Similarity
$$\text{Hybrid} = 0.6 \times \text{Content Similarity} + 0.4 \times \text{Collaborative Similarity}$$

Hybrid=0.6×Content Similarity+0.4×Collaborative Similarity

Q: How does the `recommend()` function work?

A: It finds the movie's ID, retrieves its similarity scores from the hybrid matrix, sorts them, and displays the top N similar movies.

Q: Why use cosine similarity in both methods?

A: Because cosine similarity efficiently measures how close two vectors are regardless of magnitude, making it ideal for both textual and numerical latent features.

Q: What is the output of this code?

A: A list of top N movies that are most similar to the given movie title, using both content and rating data.

Q: What kind of problem does this system solve?

A: The cold-start problem (when a new movie lacks user ratings) and sparsity problem (when users have rated very few movies).

Q: What is the role of the `recommend("Toy Story (1995)")` call?

A: It tests the recommender by showing hybrid recommendations for the movie "Toy Story (1995)."

Q: How could this hybrid recommender be improved further?

A:

- Tune blending weights using validation.
 - Add movie descriptions, cast, or reviews.
 - Incorporate deep learning (Autoencoders) or embeddings (Word2Vec/BERT).
 - Make it user-personalized.
-

Q: Why is NMF suitable for recommendation tasks?

A: Because it decomposes the rating matrix into interpretable non-negative latent features, and can handle sparse data efficiently.

Q: What is a limitation of this simple hybrid model?

A: It uses fixed weights for blending instead of learning the optimal combination dynamically. It also doesn't consider temporal effects or user profiles directly.

Q: What is the key difference between hybrid and ensemble systems?

A: Hybrid combines different types of recommendation logic (content + collaborative), while ensemble combines multiple models of the same type for better accuracy.