Assignment No-3

Title-Develop a hybrid recommender system for movie rating applications

Objectives-

- To understand the principles of hybrid recommendation systems combining content-based and collaborative filtering techniques.
- To implement a functional hybrid recommender system using a movie dataset.
- To evaluate the effectiveness of the system in providing relevant movie recommendations.

Outcomes-Students will be able to

- Understand and apply the principles of recommendation systems using Python and relevant libraries.
- Effectively clean, preprocess, and analyze data for user-item matrices.
- Implement similarity calculations and matrix factorization algorithms while defining and interpreting evaluation metrics.
- Design and integrate a hybrid recommendation system, analyzing its impact on user experience.
- Document work, collaborate in teams, and recognize ethical considerations in recommendation systems.

Theory-

Developing a hybrid recommender system for movie rating applications combines multiple recommendation techniques to provide more accurate and diverse suggestions. A common approach is to integrate content-based filtering and collaborative filtering.

Components of a Hybrid Recommender System

1. Content-Based Filtering: This method recommends items similar to those a user has liked in the past based on their features (e.g., genre, description).
2. Collaborative Filtering: This method recommends items based on the preferences and behaviors of similar users. It can be user-based (finding users with similar tastes) or item-based (finding items that are liked by users who liked the same items).
3. Blending: A hybrid system can combine the outputs of both approaches to generate final recommendations. This can be done using various methods, such as weighted average, switching, or stacking.

Steps to Build a Hybrid Recommender System

Step 1: Load the Dataset

You can use datasets like MovieLens or your own dataset that includes user ratings, movie titles, genres, and descriptions.

Step 2: Preprocess the Data

- Clean and format the data.
- Handle missing values if any.

Step 3: Implement Content-Based Filtering

- Extract features using TF-IDF or similar techniques.
- Calculate similarity scores.

Step 4: Implement Collaborative Filtering

- Use techniques like matrix factorization (e.g., SVD, NMF) or k-nearest neighbors.

Step 5: Combine the Recommendations

- Use a blending technique to combine results from both approaches.

```
import pandas as pd
import ast
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel, pairwise_distances
from scipy.sparse import csr_matrix
from sklearn.decomposition import TruncatedSVD
import numpy as np

# Step 1: Load and clean dataset
movies = pd.read_csv('/content/tmdb_5000_movies.csv')
movies = movies[['title', 'genres', 'overview']]
movies.dropna(inplace=True)

# Convert JSON-like 'genres' column to plain text
movies['genres'] = movies['genres'].apply(lambda x: '
'.join([i['name'] for i in ast.literal_eval(x)]))
movies['content'] = movies['genres'] + ' ' + movies['overview']

# Step 2: Content-based TF-IDF similarity
tfidf = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf.fit_transform(movies['content'])
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)

# Step 3: Create a sample user-rating matrix for collaborative
filtering
np.random.seed(42)
user_ids = [1, 2, 3, 4, 5]
sample_movies = movies['title'].sample(10, random_state=42).tolist()

ratings_data = []
for user in user_ids:
    for title in np.random.choice(sample_movies, 5, replace=False):
        ratings_data.append({'user_id': user, 'title': title,
'rating': np.random.randint(3, 6)})

ratings_df = pd.DataFrame(ratings_data)

# Step 4: Collaborative filtering using SVD
user_item_matrix = ratings_df.pivot(index='user_id', columns='title',
values='rating').fillna(0)
user_item_sparse = csr_matrix(user_item_matrix.values)
svd = TruncatedSVD(n_components=2)
latent_matrix = svd.fit_transform(user_item_sparse)

# Step 5: Recommendation functions
def get_content_based_recommendations(title, cosine_sim=cosine_sim):
    if title not in movies['title'].values:
        return []
    idx = movies.index[movies['title'] == title][0]
    sim_scores = list(enumerate(cosine_sim[idx]))
```

```
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
[1:4]
    return [movies['title'].iloc[i[0]] for i in sim_scores]

def get_collaborative_recommendations(user_id):
    if user_id not in user_item_matrix.index:
        return []
    user_idx = list(user_item_matrix.index).index(user_id)
    similar_users =
pairwise_distances(latent_matrix[user_idx].reshape(1, -1),
latent_matrix, metric='cosine')[0]
    similar_users_indices = np.argsort(similar_users)[:3]
    recommended_movies = []
    for idx in similar_users_indices:
        uid = list(user_item_matrix.index)[idx]
        recommended_movies.extend(ratings_df[ratings_df['user_id'] ==
uid]['title'].tolist())
    return list(set(recommended_movies))

def hybrid_recommendations(user_id, title):
    content_based = get_content_based_recommendations(title)
    collaborative_based = get_collaborative_recommendations(user_id)
    combined = list(set(content_based + collaborative_based))
    return combined

# Step 6: Example Usage
user_id = 1
movie_title = 'The Godfather'
recommended_movies = hybrid_recommendations(user_id, movie_title)
print("Hybrid Recommendations for user {} and movie
'{}':".format(user_id, movie_title))
print(recommended_movies)
```

```
Hybrid Recommendations for user 1 and movie 'The Godfather':
[np.str_('Emma'), np.str_('I Spy'), np.str_("Who's Your Caddy?"),
np.str_('Chicago'), np.str_('Fortress'), np.str_('Meet the Parents'),
'Take the Lead', np.str_('Sleepover'), 'Step Up 2: The Streets',
np.str_('AVP: Alien vs. Predator'), 'Desert Dancer']
```