**Report: Movie Recommendation Systems**

**Introduction**

This project explores two types of recommendation systems for movies: **Item-based Collaborative Filtering** and **Content-based Filtering**. The recommendation systems are built using the MovieLens dataset, which includes ratings and movie metadata. The goal is to predict similar movies based on user preferences using both content-based and collaborative filtering techniques.

**Data**

The dataset consists of two CSV files:

* **Ratings**: Contains user ratings for movies.
* **Movies**: Contains movie metadata, including the title and genres of each movie.

The following code snippets and explanations describe how the data is used to build the recommendation systems.

**Data Preprocessing**

The datasets are merged on the movieId to associate ratings with movie titles and genres. Here's the merged dataset:

df\_merged = pd.merge(ratings, movies, on='movieId')

**Item-Based Collaborative Filtering**

**Data Grouping**

First, we group the data by movie titles and genres to calculate the average rating for each movie:

df = df\_merged.groupby(['title', 'genres'])['rating'].mean().reset\_index()

This results in a dataset where each movie has a mean rating, which will be used for generating recommendations.

**Content-Based Filtering Using TF-IDF**

In the content-based filtering approach, we focus on the genres of the movies. To process the genre information, we use **TF-IDF (Term Frequency-Inverse Document Frequency)** to vectorize the genres and compute the cosine similarity between movies based on their genre vectors.

from sklearn.feature\_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(stop\_words='english')

tfidf\_matrix = tfidf.fit\_transform(df['genres'])

The similarity between movies is calculated using the cosine similarity:

from sklearn.metrics.pairwise import linear\_kernel

cosine\_sim = linear\_kernel(tfidf\_matrix, tfidf\_matrix)

**Recommendation Function (Content-Based)**

The function recommend() generates movie recommendations based on a given movie title by finding movies with the highest cosine similarity:

def recommend(title):

idx = df.loc[df['title'] == title].index[0]

sim\_scores = list(enumerate(cosine\_sim[idx]))

sim\_scores = sorted(sim\_scores, key=lambda x: x[1], reverse=True)

sim\_scores = sim\_scores[1:11]

movie\_indices = [i[0] for i in sim\_scores]

return df['title'].iloc[movie\_indices]

Example output for the movie **"Toy Story (1995)"**:

recommend('Toy Story (1995)')

Results:

* Antz (1998)
* Asterix and the Vikings (Astérix et les Vikings) (2006)
* Emperor's New Groove, The (2000)
* Moana (2016)
* Monsters, Inc. (2001)
* Shrek the Third (2007)
* Tale of Despereaux, The (2008)
* The Good Dinosaur (2015)
* Toy Story (1995)
* Toy Story 2 (1999)

**Item-Item Collaborative Filtering**

In the item-item collaborative filtering approach, we compute the Pearson correlation matrix between items (movies). This involves creating a pivot table with users as rows and movies as columns, and then calculating the correlation matrix:

pivot\_table = ratings.pivot(index='userId', columns='movieId', values='rating')

item\_item\_similarity = pivot\_table.corr(method='pearson')

item\_item\_similarity.fillna(0, inplace=True)

A heatmap of the item-item similarity matrix is plotted:

sns.heatmap(item\_item\_similarity, cmap="inferno", annot=False)

**Recommendation Function (Item-Item)**

The recommend\_top5() function generates recommendations based on item-item similarity, by finding the most similar movies to a given movie:

def recommend\_top5(title):

movie\_id = movies.loc[movies['title'] == title, 'movieId'].values[0]

sim\_scores = item\_item\_similarity[movie\_id]

sim\_scores = sim\_scores.sort\_values(ascending=False)

top\_5\_movie\_ids = sim\_scores.index[1:6]

top\_5\_titles = movies.loc[movies['movieId'].isin(top\_5\_movie\_ids), 'title']

return top\_5\_titles

Example output for the movie **"Toy Story (1995)"**:

recommend\_top5('Toy Story (1995)')

Results:

* Child's Play 2 (1990)
* Quicksilver (1986)
* Amen. (2002)
* Bachelor and the Bobby-Soxer, The (1947)
* Date Movie (2006)

**User Similarity**

In addition to item-based collaborative filtering, we can also compute **user-user similarity** to generate recommendations based on similar users. This method involves computing the Pearson correlation matrix for users based on their ratings of movies.

**User-User Collaborative Filtering**

First, we create a pivot table with users as rows and movies as columns, and ratings as values:

user\_pivot\_table = ratings.pivot(index='userId', columns='movieId', values='rating')

Next, we compute the Pearson correlation matrix for users:

user\_user\_similarity = user\_pivot\_table.T.corr(method='pearson')

user\_user\_similarity.fillna(0, inplace=True)

The matrix is filled with values that represent the similarity between different users based on their ratings of movies.

**Finding Top-N Similar Users**

The function find\_top\_similar\_users() allows us to find the most similar users to a given user by comparing the similarity scores:

def find\_top\_similar\_users(user\_id, top\_n=5):

# Get the similarity scores for the given user

sim\_scores = user\_user\_similarity[user\_id]

# Sort the similarity scores in descending order

sim\_scores = sim\_scores.sort\_values(ascending=False)

# Get the top-n similar userIds

top\_n\_user\_ids = sim\_scores.index[1:top\_n+1]

return top\_n\_user\_ids

For example, using the function with user\_id = 1:

find\_top\_similar\_users(1)

This function will return the top 5 users that are most similar to user 1, based on their ratings of movies.

**Evaluation**

To evaluate the recommendation systems, we would typically use metrics such as **Mean Absolute Error (MAE)**, **Root Mean Squared Error (RMSE)**, or **Precision and Recall**. However, in this case, there was no separate test dataset available to evaluate the model effectively.

**Conclusion**

In this project, two recommendation systems were built:

1. **Content-Based Filtering**: Utilizes movie genres and computes similarity using TF-IDF and cosine similarity.
2. **Collaborative Filtering**: Uses user-item interactions (ratings) and computes similarity between items (movies) using Pearson correlation.

Both approaches provide movie recommendations, and the content-based filtering approach offers a more accurate recommendation for movies with similar genres. However, item-item collaborative filtering can be particularly useful when dealing with large datasets and user-item interactions.

Heat map for movie to movie similarity:



