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.

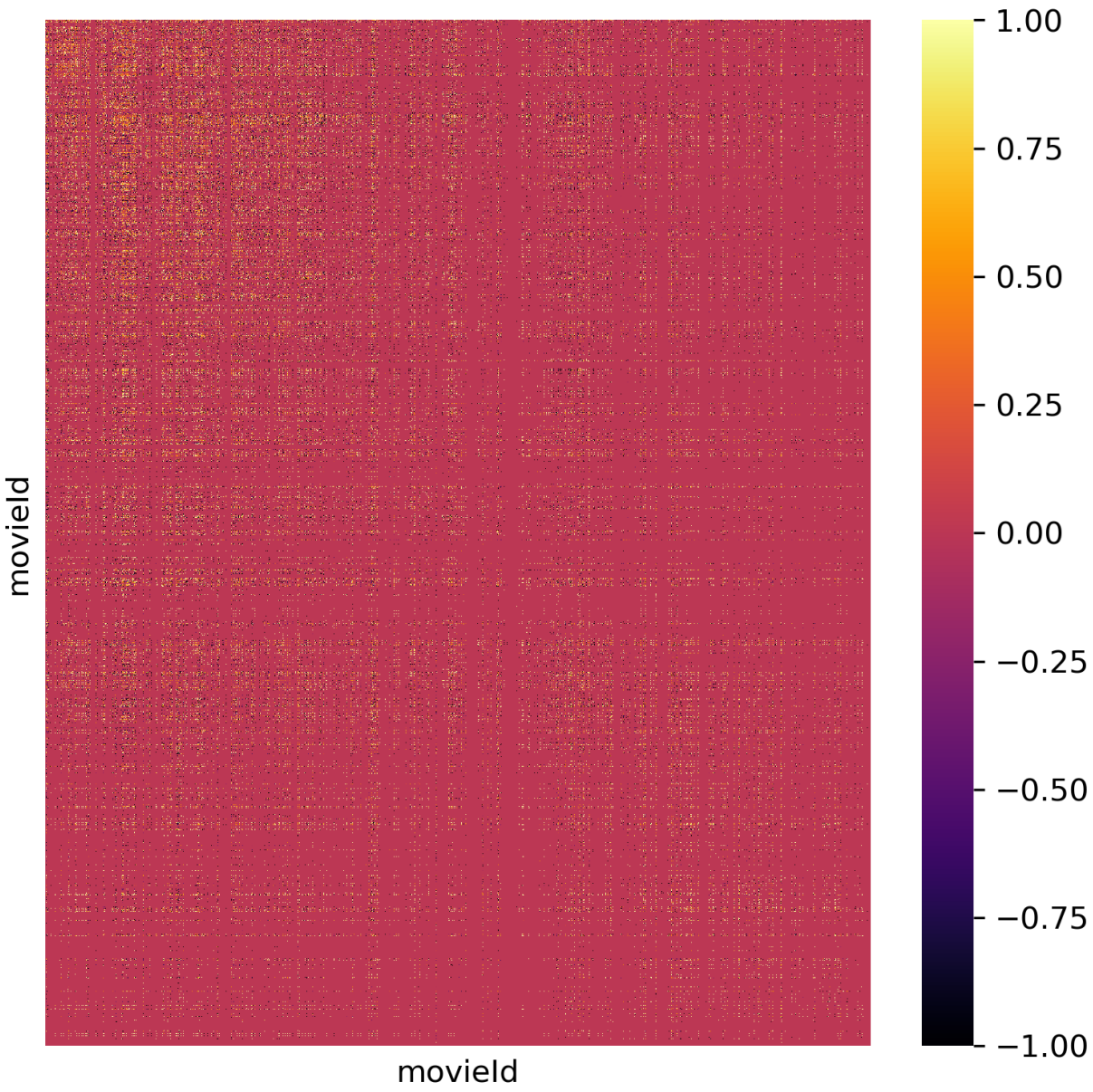
Conclusion

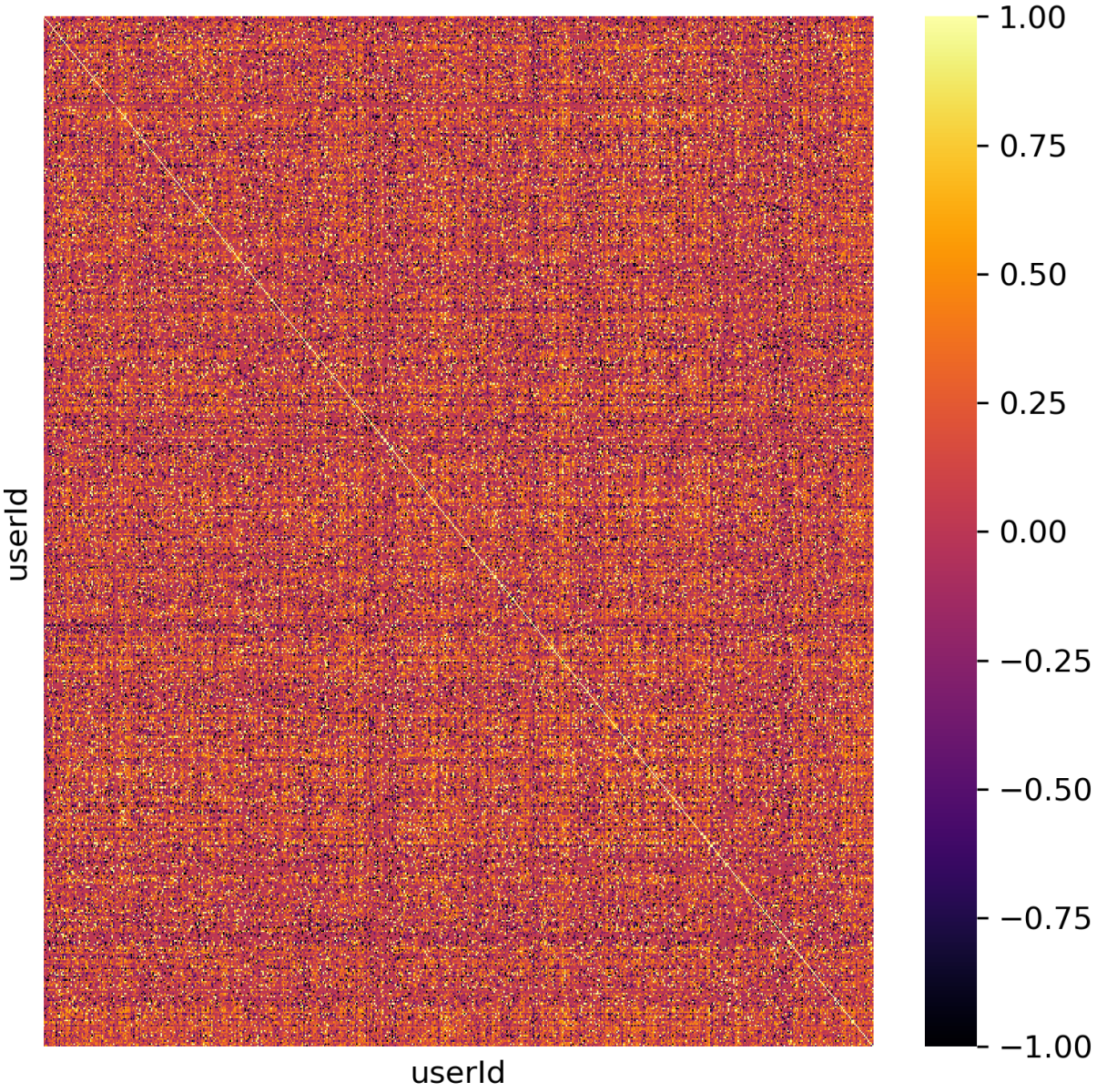
In this project, two recommendation systems were built:

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:




userId
userId
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00