Design and Implementation of a Hybrid Movie Recommendation System

Final Thesis Report

Submitted by

Rohit Andani

Introduction to Artificial Intelligence - OL-IN Fall 2024 B

Student of

DECEMBER 2024

DEDICATION

I dedicate this thesis to my family and mentors. To my family, thank you for your endless support, patience, and encouragement. Your belief in me has been my greatest motivation. To my mentors, I am deeply grateful for your guidance and wisdom, which have shaped my understanding and growth throughout this journey. This work is a testament to your unwaveringsupport.

ABSTRACT

This thesis addresses to build a comprehensive movie recommendation system using collaborative filtering, content-based filtering, and hybrid approaches to deliver personalized movie suggestions to users. Recommender systems play a crucial role in enhancing user experience by suggesting content that aligns with their preferences. Utilizing the MovieLens dataset, this project involves data loading, preprocessing, similarity calculations, and applying various filtering algorithms to develop an effective recommendation system.

Three primary recommendation functions are developed in this project: user-based collaborative filtering, item-based collaborative filtering, and content-based filtering.

User-Based Collaborative Filtering: This method identifies users with similar preferences to the target user and recommends movies that these similar users have liked. The algorithm calculates the weighted average of ratings from similar users to generate recommendations.

Item-Based Collaborative Filtering: This approach finds movies similar to those the target user has rated highly and recommends these similar movies. The algorithm relies on item-item similarity to identify and suggest movies.

Content-Based Filtering: This technique uses movie attributes, such as genres, to recommend movies similar to the ones the user has liked. The algorithm employs TF-IDF vectorization and cosine similarity to compute content similarity between movies.

The project successfully demonstrates the implementation of a movie recommendation system using collaborative filtering, content-based filtering, and hybrid approaches. Each method has its strengths and limitations, and the hybrid approach provides a comprehensive solution by combining the best aspects of all techniques. Future work can focus on further enhancing the hybrid model and addressing challenges such as the cold start problem and scalability.

Table of Contents

LIST OF FIGURES

1. Introduction

The rapid growth of online content platforms has necessitated the development of robust recommendation systems to enhance user experience. This project implements a hybrid movie recommendation system leveraging collaborative filtering, content-based filtering, and hybrid methodologies. Using the MovieLens dataset, this study demonstrates the design and implementation of algorithms for user-item interaction analysis, cosine similarity computation, and TF-IDF vectorization. The results indicate that hybrid models can significantly improve recommendation accuracy by combining strengths of individual techniques.

Recommendation systems have become integral to digital platforms, offering personalized experiences by predicting user preferences. This project explores three primary recommendation techniques—collaborative filtering, content-based filtering, and hybrid approaches—using the MovieLens dataset. The study emphasizes practical implementations, visualization, and evaluation of these techniques.

1.1 Brief Discussion of the Problem

The primary goal of this project is to build a robust movie recommendation system using a combination of collaborative filtering, content-based filtering, and a hybrid approach. Movie recommendation systems play a crucial role in enhancing user experience by suggesting movies that align with user preferences and interests. This system will analyze user ratings, movie attributes, and user-item interactions to provide personalized recommendations.

1.2 Algorithms Investigated

1. **User-Based Collaborative Filtering**: Identifies users with similar preferences to the target user and recommends movies that these similar users liked.
2. **Item-Based Collaborative Filtering**: Finds movies similar to those the target user has rated highly and recommends those.
3. **Content-Based Filtering**: Utilizes the attributes of movies (genres and tags) to recommend movies similar to the ones the user has liked.
4. **Hybrid Approach**: Combines the strengths of user-based, item-based, and content-based filtering to improve recommendation accuracy and address the limitations of each individual method.

1.3 Framework Development

1. **Data Loading and Preprocessing**:
    - **Load Datasets**: The system loads user ratings and movie metadata from CSV files.
    - **Preprocess Data**: The data is inspected and transformed into a user-item matrix. Content features such as movie genres are extracted and processed.
    - **User-Item Matrix**: Created from user ratings, where rows represent users, columns represent movies, and values represent ratings.
    - **Content Features**: Movie attributes like genres are processed for content-based filtering.
2. **Similarity Calculations**:
    - **User Similarity**: Calculated using cosine similarity to find users with similar rating patterns.
    - **Item Similarity**: Calculated using cosine similarity to find movies with similar rating patterns.
3. **Recommendation Algorithms**:
    - **User-Based Filtering**: Recommends movies based on the preferences of similar users.
    - **Item-Based Filtering**: Recommends movies similar to those the user has rated highly.
    - **Content-Based Filtering**: Recommends movies based on their attributes.
    - **Hybrid Approach**: Combines multiple filtering methods to enhance recommendation accuracy.
4. **Recommendations Output**:
    - **Display Recommendations**: The system presents the recommended movies to the user based on the selected algorithm.

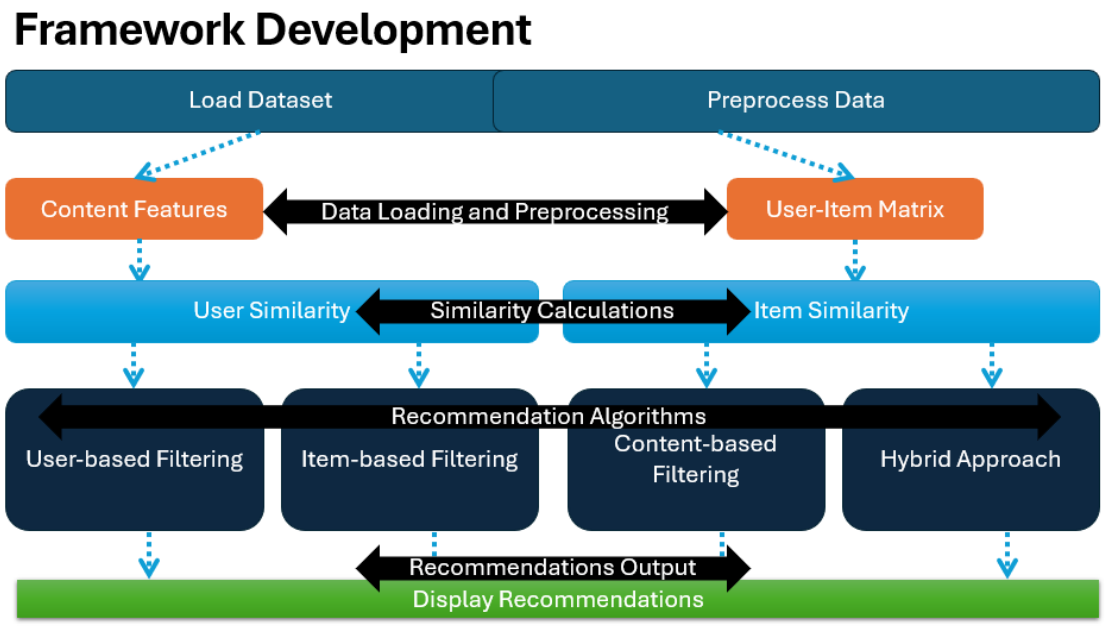


Figure 1 Framework Development Diagram

1.4 Research Questions

This thesis addresses specific use-case around the movie recommendation system based on various algorithms and behavior analysis with focus on the AI/ML adoption to capture the right set of suggestions to improve platform relevance.

The research questions guiding this study are:

1. How effective are collaborative filtering techniques in providing personalized movie recommendations?
2. What are the limitations of content-based filtering in movie recommendation systems, and how can they be mitigated?
3. How can a hybrid recommendation approach improve the accuracy and robustness of movie recommendations?
4. What is the impact of data preprocessing and similarity calculations on the overall performance of the recommendation system?
5. How can visualization techniques aid in understanding user behavior and

the performance of recommendation algorithms?

6. How do the implemented recommendation algorithms compare in terms of computational efficiency and scalability?
7. What are the best practices for evaluating the performance of recommendation systems?

2. Data Loading and Preprocessing

Data preprocessing is critical to the success of recommendation systems. The MovieLens dataset, consisting of user ratings and movie metadata, is loaded and cleaned. A user-item interaction matrix is created, and missing values are handled appropriately to facilitate similarity calculations.

The system is built using Python and libraries such as pandas, sklearn, matplotlib, seaborn, and Plotly. It will include:

- Data loading and preprocessing modules.
- Functions for calculating user and item similarities.
- Recommendation algorithms for user-based, item-based, content-based, and hybrid methods.
- Visualization components to provide insights into user-item interactions and recommendation performance.
- A web interface using Flask for interacting with the recommendation system.
- An interactive dashboard using Dash for visualizing data and recommendations.

The first step in building the recommendation system is loading the datasets and preprocessing them for analysis. We use the MovieLens dataset, which contains user ratings and movie information. The preprocessing steps include inspecting the datasets and creating a user-item matrix.

1. Data Loading Functionality: This part loads the MovieLens datasets from local files.

```python
import pandas as pd

# Load datasets
def load_datasets():
    ratings = pd.read_csv('C:/Users/rohit/Downloads/ml-latest-small/ml-latest-small/ratings.csv')
    movies = pd.read_csv('C:/Users/rohit/Downloads/ml-latest-small/ml-latest-small/movies.csv')
    return ratings, movies

# Inspect datasets
def inspect_datasets(ratings, movies):
    print(ratings.head())
    print(movies.head())

# Create user-item matrix
def create_user_item_matrix(ratings):
    user_item_matrix = ratings.pivot(index='userId', columns='movieId', values='rating').fillna(0)
    return user_item_matrix

# Usage
ratings, movies = load_datasets()
inspect_datasets(ratings, movies)
user_item_matrix = create_user_item_matrix(ratings)
print(user_item_matrix.head())
```

[15]

```
   userId  movieId  rating  timestamp
0       1        1     4.0  964982703
1       1        3     4.0  964981247
2       1        6     4.0  964982224
3       1       47     5.0  964983815
4       1       50     5.0  964982931
   movieId                               title  \
0        1                    Toy Story (1995)
1        2                      Jumanji (1995)
2        3             Grumpier Old Men (1995)
3        4            Waiting to Exhale (1995)
4        5  Father of the Bride Part II (1995)

                                        genres
0  Adventure|Animation|Children|Comedy|Fantasy
1                   Adventure|Children|Fantasy
2                               Comedy|Romance
3                         Comedy|Drama|Romance
4                                       Comedy
movieId  1      2      3      4      5      6      7      8      \
userId
1          4.0    0.0    4.0    0.0    0.0    4.0    0.0    0.0
2          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
3          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
4          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5          4.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0

movieId  9      10     ...  193565  193567  193571  193573  193579  193581  \
userId                 ...
1          0.0    0.0  ...     0.0     0.0     0.0     0.0     0.0     0.0
```

2.1 Visualizing the Data

Data visualization techniques, such as heatmaps, are employed to explore user-item interactions and rating distributions. These visualizations offer insights into user behavior patterns and the sparsity of the dataset. Visualization is crucial for understanding the data and its distribution. This section includes:

- **User-Item Interaction Heatmap**: Visualizing the user-item matrix using a heatmap to see the interaction patterns.

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Visualize User-Item Interaction Heatmap
def visualize_user_item_heatmap(user_item_matrix):
    plt.figure(figsize=(10, 8))
    sns.heatmap(user_item_matrix, cmap='viridis', cbar=False)
    plt.title('User-Item Interaction Heatmap')
    plt.xlabel('Movie ID')
    plt.ylabel('User ID')
    plt.show()

# Visualize Distribution of Ratings
def visualize_ratings_distribution(ratings):
    plt.figure(figsize=(8, 6))
    ratings['rating'].hist(bins=10)
    plt.title('Distribution of Ratings')
    plt.xlabel('Rating')
    plt.ylabel('Count')
    plt.show()

# Usage
visualize_user_item_heatmap(user_item_matrix)
visualize_ratings_distribution(ratings)
```

Key Observations:

1. **X-Axis (Movie ID)**:
    - The x-axis represents movie IDs, ranging from 1 to 14681. Each column corresponds to a specific movie.
2. **Y-Axis (User ID)**:
    - The y-axis represents user IDs, ranging from 1 to 603. Each row corresponds to a specific user.
3. **Color Intensity**:
    - Each cell in the heatmap indicates whether a user has interacted with a particular movie.
    - **Darker colors** represent fewer interactions or no interactions.
    - **Lighter colors** represent more interactions or stronger interactions.
4. **Patterns**:
    - The heatmap allows us to identify patterns in user behavior. For example, certain movies might have many interactions (light-colored columns), while others have few (dark-colored columns).
    - Similarly, some users might interact with many movies (lighter rows), while others interact with few (darker rows).

Interpretation:

1. **User Behavior**:

- Users who frequently rate movies will have lighter rows, indicating higher interaction levels.
- Users who rarely rate movies will have darker rows.

2. **Movie Popularity**:

   - Movies with high interaction rates will have lighter columns, indicating they are popular among users.
   - Movies with low interaction rates will have darker columns, indicating they are less popular.

3. **Recommendation Insights**:

   - The heatmap helps in understanding which users are more active and which movies are more popular. This information is crucial for collaborative filtering systems to make accurate recommendations.

   - It can also highlight gaps, such as users who need more recommendations or movies that are underexplored by the user base.

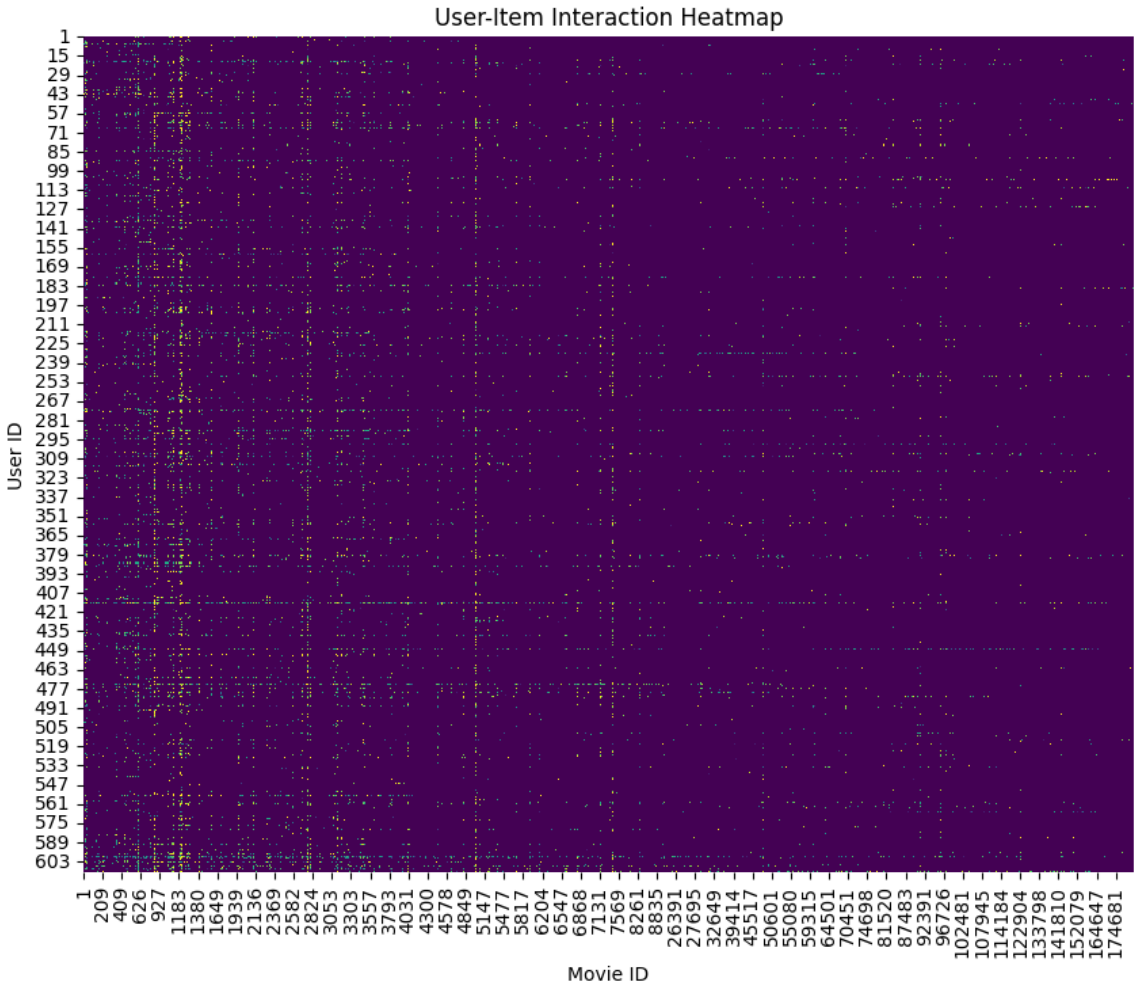


Figure 2 Use-item Interaction Heatmap

- **Distribution of Ratings**: Plotting the distribution of movie ratings to understand the rating behavior of users.

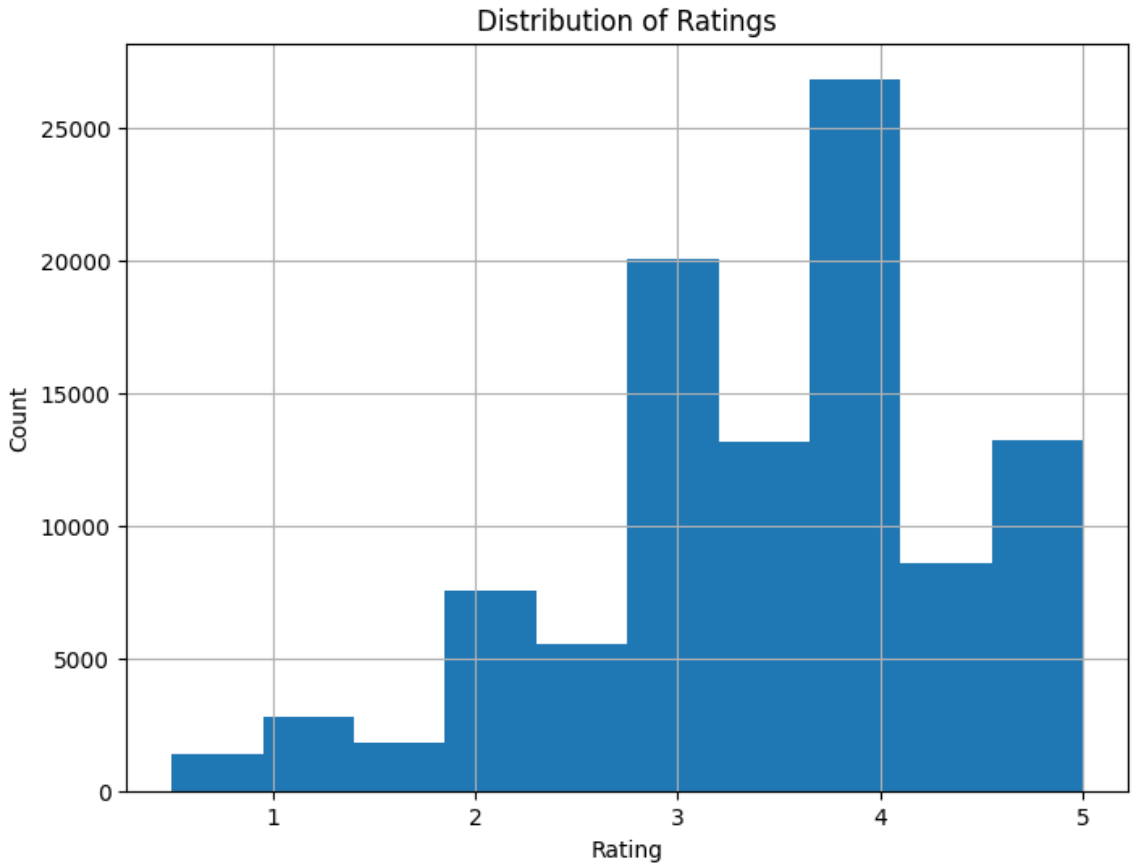


Figure 3 Distribution of Ratings

Key Observations:

- **X-Axis (Ratings)**:

  The x-axis represents the rating values, ranging from 1 to 5. These values indicate how users rated various movies.

- **Y-Axis (Frequency)**:

  The y-axis represents the count of each rating, showing how many times each rating value was given by users.

Rating Distribution:

- **Rating of 4**: The highest count, with over 25,000 occurrences. This indicates that many users rated movies with a score of 4.
- **Rating of 3**: The second highest count, with over 20,000 occurrences. Many users gave a rating of 3.
- **Rating of 5**: This rating has around 15,000 occurrences, suggesting a significant number of users gave the highest possible rating.
- **Rating of 2**: This rating has around 10,000 occurrences, indicating fewer users gave a rating of 2.
- **Rating of 1**: The lowest count, with below 5,000 occurrences, showing that very few users gave the lowest rating.

Interpretation:

- ✓ **Skewed Distribution**: The histogram shows that higher ratings (3, 4, and 5) are more common than lower ratings (1 and 2). This suggests that users tend to rate movies more positively.
- ✓ **User Preferences**: Most users prefer to rate movies with a score of 3 or higher, indicating overall satisfaction with the movies they watched.
- ✓ **Bias Towards Higher Ratings**: The skew towards higher ratings may indicate a positive bias among users, where they are more likely to rate movies favorably.

3. Similarity Calculations

In a movie recommendation system, understanding the similarity between users or items is essential for making accurate recommendations. Similarity calculations help in identifying users with similar tastes or items that are alike. This process is the foundation of collaborative filtering techniques, both user-based and item-based. The most common method used for calculating similarity is cosine similarity, which measures the cosine of the angle between two non-zero vectors in a multi-dimensional space. This narration will explore the steps involved in calculating similarity, its importance, and how it is visualized and used in recommendation systems.

Cosine similarity is a key metric used in this project to determine the likeness between users or items. It measures the cosine of the angle between two non-zero vectors, making it suitable for high-dimensional, sparse datasets like the MovieLens ratings.

Similarity calculations are a fundamental part of building effective recommendation systems. By understanding the relationships between users and items, these calculations enable personalized and accurate recommendations. Implementing cosine similarity using user-item matrices and visualizing the results with heatmaps provides valuable insights into user behavior and item characteristics. Despite challenges such as data sparsity and scalability, similarity calculations remain a powerful tool in the development of recommendation systems, enhancing user experience and engagement.

Cosine Similarity:

- Cosine similarity calculates the cosine of the angle between two vectors, representing two users' rating patterns. The formula for cosine similarity between two vectors A and B is: (A . B) / (||A|| ||B||).
- This results in a value between -1 and 1, where 1 indicates identical preferences, -1 indicates completely opposite preferences, and 0 indicates no similarity.

```
from sklearn.metrics.pairwise import cosine_similarity

# Calculate User Similarity
def calculate_user_similarity(user_item_matrix):
    user_similarity = cosine_similarity(user_item_matrix)
    user_similarity_df = pd.DataFrame(user_similarity, index=user_item_matrix.index, columns=user_item_matrix.index)
    return user_similarity_df

# Calculate Item Similarity
def calculate_item_similarity(user_item_matrix):
    item_similarity = cosine_similarity(user_item_matrix.T)
    item_similarity_df = pd.DataFrame(item_similarity, index=user_item_matrix.columns, columns=user_item_matrix.columns)
    return item_similarity_df

# Usage
user_similarity_df = calculate_user_similarity(user_item_matrix)
item_similarity_df = calculate_item_similarity(user_item_matrix)
print(user_similarity_df.head())
print(item_similarity_df.head())
# Visualize User Similarity Matrix
def visualize_user_similarity_heatmap(user_similarity_df):
    plt.figure(figsize=(10, 8))
    sns.heatmap(user_similarity_df, cmap='coolwarm', annot=False, fmt='.2f')
    plt.title('User Similarity Matrix')
    plt.xlabel('User ID')
    plt.ylabel('User ID')
    plt.show()

# Visualize Item Similarity Matrix
def visualize_item_similarity_heatmap(item_similarity_df):
    plt.figure(figsize=(10, 8))
    sns.heatmap(item_similarity_df, cmap='coolwarm', annot=False, fmt='.2f')
    plt.title('Item Similarity Matrix')
    plt.xlabel('Movie ID')
    plt.ylabel('Movie ID')
    plt.show()

# Usage
visualize_user_similarity_heatmap(user_similarity_df)
visualize_item_similarity_heatmap(item_similarity_df)
```

3.1 User Similarity Calculation:

Using cosine similarity to compute the similarity between users based on their ratings. User similarity calculation is a critical component of User-Based Collaborative Filtering (UBCF). This process involves determining how similar users are to each other based on their ratings of items (e.g., movies). The similarity scores are then used to generate personalized recommendations.

User similarity calculation is essential for identifying users with similar preferences in User-Based Collaborative Filtering. By using cosine similarity, we can quantify the degree of similarity between users based on their ratings. The resulting user similarity matrix forms the foundation for generating personalized recommendations, enhancing the user experience by suggesting items that users with similar tastes have enjoyed

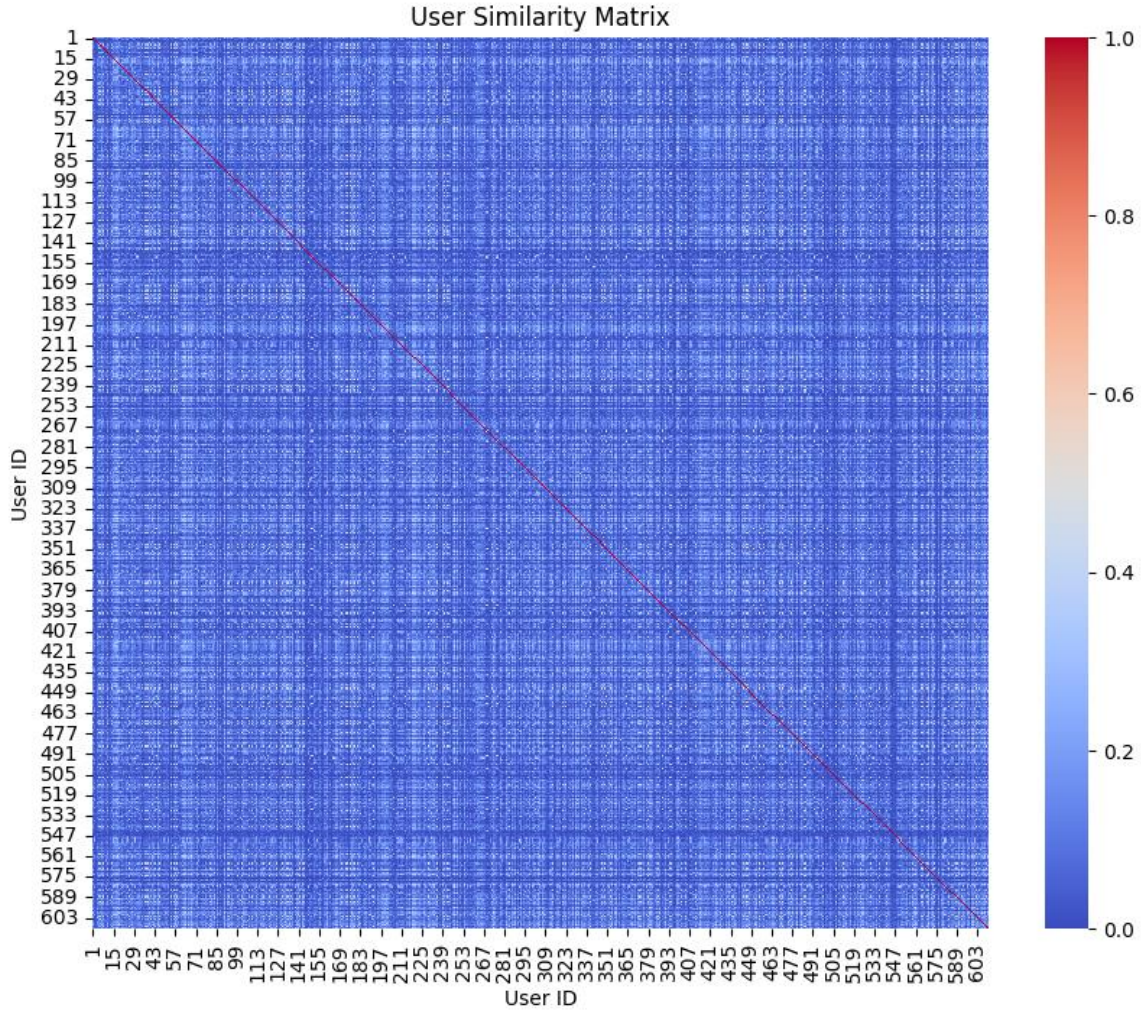


Figure 4 user Similarity Matrix

Key Observations:

1. **Diagonal Line**:

- The diagonal from the top-left to the bottom-right is red, indicating that each user is perfectly similar to themselves, with a similarity score of 1. This is expected, as any user's similarity with themselves should be the maximum.

2. **Color Scale**:
   - The color scale on the right ranges from blue (low similarity) to red (high similarity), with a gradient in between. This helps in visually distinguishing the degrees of similarity between users.
3. **Low Similarity**:
   - Most of the off-diagonal cells are blue, suggesting low similarity between different users. This means that most users have unique movie preferences that do not closely align with those of others.
4. **Varying Similarity**:
   - There are variations in the shades of blue, indicating varying degrees of similarity among different user pairs. For instance, some users might have moderate similarity, represented by lighter shades of blue or intermediary colors.

3.2 Item Similarity Calculation:

Using cosine similarity to compute the similarity between movies based on user ratings. The cosine similarity between items is calculated similarly to users, but it measures the similarity of the rating patterns for different movies. Item similarity calculation is a crucial part of Item-Based Collaborative Filtering (IBCF). This process involves determining how similar items (e.g., movies) are to each other based on user ratings. The similarity scores are then used to

generate recommendations for users based on items they have liked.

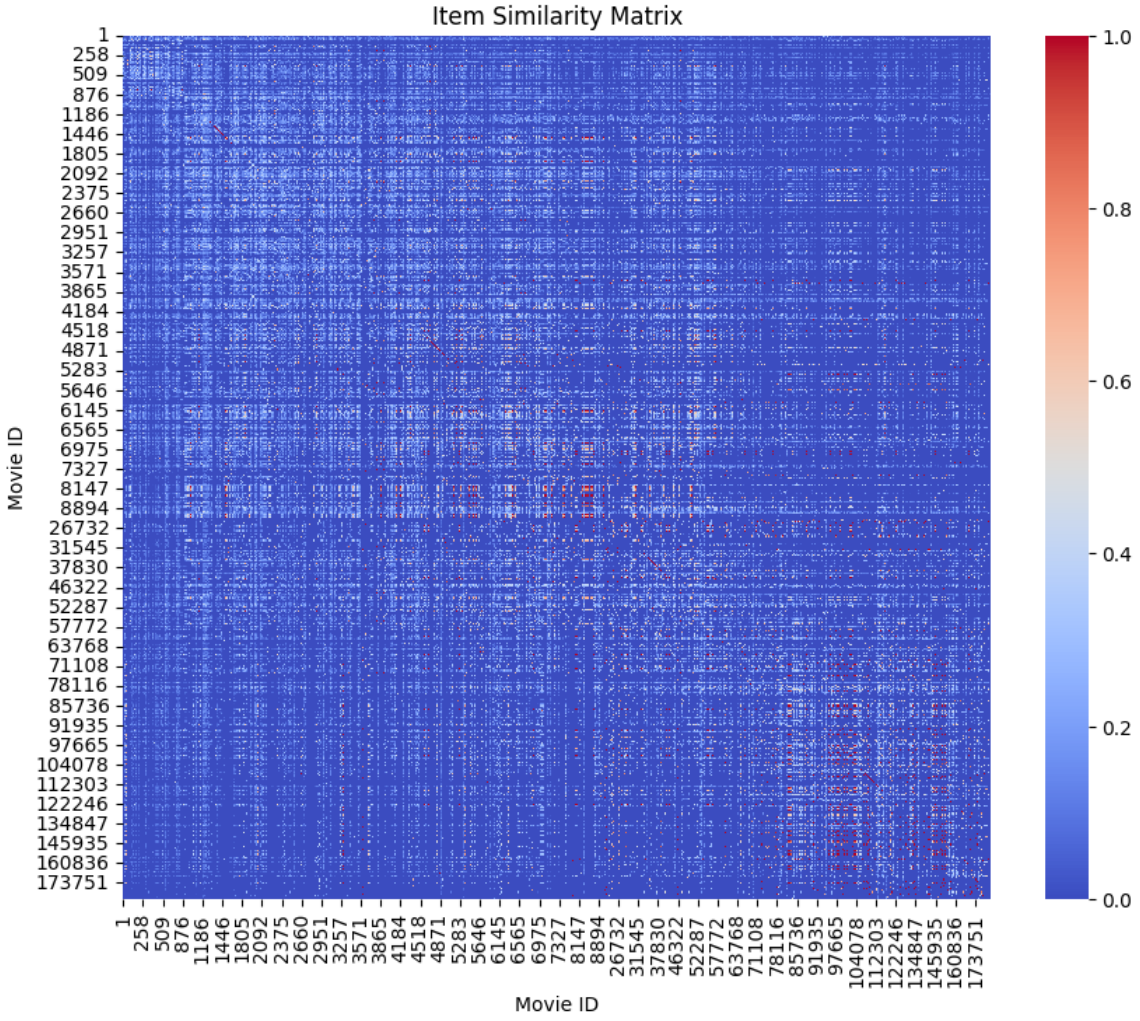


Figure 5 item Similarity Matrix

Key Observations:

1. **Color Scale**:
    - Similar to the user similarity matrix, the color scale ranges from blue (low similarity) to red (high similarity). This helps in visually distinguishing the degrees of similarity between movies.
2. **Low Similarity**:

- The majority of the matrix is blue, indicating that most movies have low similarity with each other. This suggests that the movies in the dataset have diverse content and appeal to different user preferences.

3. **High Similarity Spots**:

    - There are some spots with red and lighter blue colors, indicating higher similarity between certain pairs of movies. These spots represent movies that share similar characteristics and have been rated similarly by users.

4. Recommendation Functions

The recommendation functions are designed to identify items of interest based on calculated similarities. These functions form the backbone of the collaborative, content-based, and hybrid filtering approaches. This section outlines the core recommendation functions that form the basis of the recommendation system:

- **User-Based Collaborative Filtering Function**: Recommending movies to users based on ratings from similar users.
- **Item-Based Collaborative Filtering Function**: Recommending movies to users based on similar movies they have rated highly.
- **Content-Based Filtering Function**: Recommending movies based on their attributes (genres).

4.1 User-Based Collaborative Filtering

User-based collaborative filtering suggests items by identifying similar users based on their rating histories. This approach assumes that users with similar preferences in the past will continue to exhibit similar behaviors. User-Based Collaborative Filtering is a powerful technique for generating personalized recommendations by leveraging the similarity between users. By understanding how users rate items similarly, the system can recommend items that similar users have liked. Despite some challenges, such as data sparsity and the cold start problem, UBCF remains a popular and effective method in the field of recommendation systems. This section dives deeper into the user-based collaborative filtering algorithm:

User-Based Collaborative Filtering (UBCF) is a recommendation technique that leverages the preferences of similar users to generate recommendations for a target user. This method operates on the premise that users who shared similar tastes in the past are likely to agree on items in the future.

- **User Similarity Computation**: Determine the similarity between users using metrics like cosine similarity, Pearson correlation, or Jaccard similarity. This step generates a user-user similarity matrix.
- **Neighbor Identification**: For a target user, identify the most similar users (neighbors) based on the similarity matrix.
- **Aggregation of Preferences**: Use the preferences of these neighbors to predict the ratings of items that the target user hasn't interacted with. Predictions are weighted by the similarity scores of the neighbors.
- **Recommendation**: Recommend items to the user that have the highest predicted ratings.

Code Implementation: Code implementation of the user-based collaborative filtering function.

```python
# User-Based Collaborative Filtering
def get_user_based_recommendations(user_id, num_recommendations, user_item_matrix, user_similarity_df):
    user_ratings = user_item_matrix.loc[user_id]
    similar_users = user_similarity_df[user_id].sort_values(ascending=False)
    recommendations = {}

    for similar_user, similarity in similar_users.items():
        if similar_user != user_id:
            similar_user_ratings = user_item_matrix.loc[similar_user]
            for movie_id, rating in similar_user_ratings.items():
                if user_ratings[movie_id] == 0:
                    if movie_id not in recommendations:
                        recommendations[movie_id] = rating * similarity
                    else:
                        recommendations[movie_id] += rating * similarity
    return sorted(recommendations.items(), key=lambda x: x[1], reverse=True)[:num_recommendations]

# Usage
user_id = 1
recommendations = get_user_based_recommendations(user_id, 5, user_item_matrix, user_similarity_df)
print("User-Based Recommendations:", recommendations)
```

[7] ✓ 8.0s

Visualization: Visualizing the recommendations provided by the algorithm.

Key Insights:

Explanation of the Output

Input Parameters:

- user_id = 1: The ID of the target user for whom we are generating recommendations.
- num_recommendations = 5: The number of movie recommendations to generate.
- user_item_matrix: The user-item matrix containing the ratings given by users to movies.
- user_similarity_df: The user similarity matrix containing similarity scores between users.

Output Parameters

- **Movie 318** has the highest score (215.45), indicating it is the most strongly recommended movie for the target user.
- The subsequent movies (589, 858, 2762, and 4993) have progressively lower scores, indicating they are less strongly recommended but still relevant.

```
User-Based Recommendations: [(318, 215.4497028920212), (589, 169.4011820030674), (858, 150.91532685877024), (2762, 135.00986370803545), (4993, 131.86554086066508)]
```

Interpretation:

- The function has identified the top 5 movies to recommend to user_id = 1 based on the ratings of similar users.
- The higher the weighted score, the more likely the target user (user_id = 1) will enjoy the movie based on the preferences of similar users.

- The recommended movies are listed in descending order of their scores, with movie ID 318 having the highest score and thus the highest recommendation priority.

4.2 Item-Based Collaborative Filtering

Item-based collaborative filtering focuses on finding items similar to those a user has already rated positively. It calculates item-item similarities to recommend items that align closely with a user's preferences. IBCF focuses on the relationships between items. If two items are frequently interacted with (e.g.viewed) together by many users, they are considered similar. For a target user, recommendations are generated by analyzing their history of interactions and suggesting items similar to those they have interacted with positively.

1. Item Similarity Computation

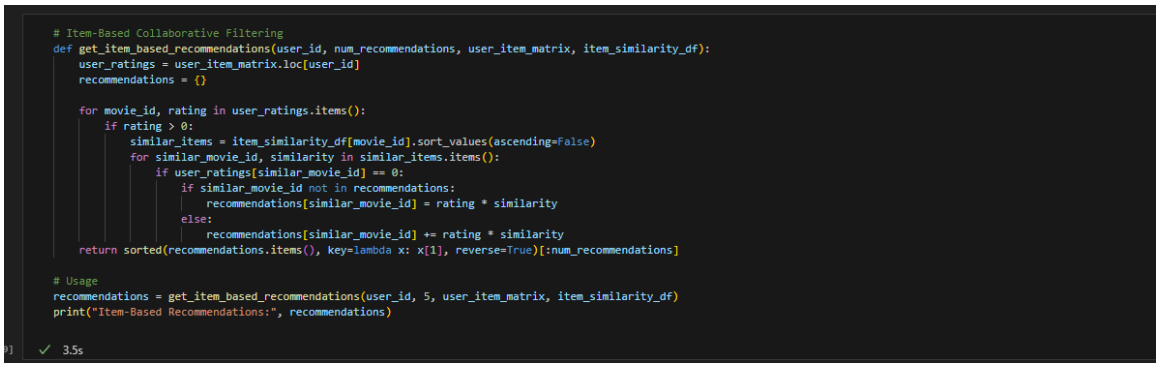
- A similarity measure (e.g., cosine similarity, Pearson correlation, or adjusted cosine similarity) is used to calculate the similarity between items.
- This results in an item-item similarity matrix, where each entry represents the degree of similarity between a pair of items.

2. Prediction Generation

- For a given user, the algorithm predicts their rating for an item by analyzing their ratings for similar items.
- The predicted rating is typically a weighted average of the user's ratings for similar items, weighted by the similarity scores.

3. Recommendation

- Items with the highest predicted ratings that the user has not interacted with are recommended.



```
# Item-Based Collaborative Filtering
def get_item_based_recommendations(user_id, num_recommendations, user_item_matrix, item_similarity_df):
    user_ratings = user_item_matrix.loc[user_id]
    recommendations = {}

    for movie_id, rating in user_ratings.items():
        if rating > 0:
            similar_items = item_similarity_df[movie_id].sort_values(ascending=False)
            for similar_movie_id, similarity in similar_items.items():
                if user_ratings[similar_movie_id] == 0:
                    if similar_movie_id not in recommendations:
                        recommendations[similar_movie_id] = rating * similarity
                    else:
                        recommendations[similar_movie_id] += rating * similarity
    return sorted(recommendations.items(), key=lambda x: x[1], reverse=True)[:num_recommendations]

# Usage
recommendations = get_item_based_recommendations(user_id, 5, user_item_matrix, item_similarity_df)
print("Item-Based Recommendations:", recommendations)

✓ 3.5s
```

Output

Input Parameters:

- user_id: The ID of the target user for whom we are generating recommendations.
- num_recommendations: The number of movie recommendations to generate.
- user_item_matrix: The user-item matrix containing the ratings given by users to movies.
- item_similarity_df: The item similarity matrix containing similarity scores between movies.

```
...  Item-Based Recommendations: [(2918, 352.6840101334495), (1036, 330.9498844896122), (1968, 328.18050722272784), (1527, 326.43737393217197), (1200, 323.1176482769818)]
```

Explanation:

- The output is a list of movie recommendations for the target user, along with their computed scores.
- **Movie ID 2918**: Recommended movie with the highest score of 352.684. This score is derived from the ratings given by the user to similar movies and their similarity scores.
- **Top 5 Movies**: The list shows the top 5 movies with the highest recommendation scores, indicating they are the most likely to be enjoyed by the user based on their past ratings.

Comparison: IBCF vs. UBCF

Aspect	IBCF	UBCF
Focus	Item similarity	User similarity
Data Dependence	Stable relationships between items	Dynamic user preferences
Cold-Start	Affects new items	Affects new users
Scalability	Efficient with many users	Challenging with a large user base
Interpretability	"Because you liked this, try that"	"Users like you also liked this"

Figure 6 Comparison of IBCF vs UBCF

4.3 Content-Based Filtering

Content-Based Filtering (CBF) is a recommendation technique that generates personalized recommendations for users by analyzing the characteristics (content) of items and comparing them to the user's past preferences. It is widely used in recommendation systems across various domains, including e-commerce, movies, music, and more. This approach focuses solely on the relationship between the user and the items, without considering other users

Content-based filtering utilizes the attributes of items—in this case, movie genres—encoded as TF-IDF vectors. Movies with similar genre profiles are recommended to users based on their previously rated movies.

Steps in Content-Based Filtering

1. **Item Profiling**:
    - Each item is represented by a set of attributes or features that describe it. For example:
        - Movies: Genre, director, cast, release year, etc.
        - Books: Author, genre, publication year, keywords, etc.
        - Products: Brand, category, specifications, price, etc.
    - These attributes form the basis of item comparison.
2. **User Profiling**:
    - A profile is created for each user by analyzing the features of items they have interacted with. For example:
        - A user who frequently rates action and sci-fi movies highly might have a profile that emphasizes these genres.
    - The user profile is typically a weighted vector of attributes, where weights indicate the user's level of interest in each feature.
3. **Similarity Computation**:
    - The similarity between the user's profile and each item is calculated using measures like:
        - **Cosine Similarity**: Measures the cosine of the angle between two vectors.
        - **Dot Product**: Computes the weighted sum of matching features.
    - Items with higher similarity scores are considered more relevant to the user.
4. **Recommendation**:

- Items with the highest similarity scores that the user has not yet interacted with are recommended.

```
# Content-Based Filtering
def get_content_based_recommendations(movie_id, num_recommendations, movies, cosine_sim):
    movie_index = movies[movies['movieId'] == movie_id].index[0]
    similarity_scores = list(enumerate(cosine_sim[movie_index]))
    similarity_scores = sorted(similarity_scores, key=lambda x: x[1], reverse=True)
    similarity_scores = similarity_scores[1:num_recommendations+1]
    movie_indices = [i[0] for i in similarity_scores]
    return movies.iloc[movie_indices]

# Usage
movie_id = 1
recommendations = get_content_based_recommendations(movie_id, 5, movies, cosine_sim)
print("Content-Based Recommendations:")
print(recommendations)

✓ 0.0s
```

Input Parameters:

- movie_id**:** The ID of the movie for which we are generating similar movie recommendations.
- num_recommendations**:** The number of movie recommendations to generate.
- movies**:** A DataFrame containing the movie metadata (like movie IDs, titles, genres).
- cosine_sim**:** The cosine similarity matrix for the movies based on their content features (like genres).

```
Content-Based Recommendations:
      movieId                                    title  \
1706     2294                                Antz (1998)
2355     3114                          Toy Story 2 (1999)
2809     3754  Adventures of Rocky and Bullwinkle, The (2000)
3000     4016           Emperor's New Groove, The (2000)
3568     4886                    Monsters, Inc. (2001)

                                      genres
1706  Adventure|Animation|Children|Comedy|Fantasy
2355  Adventure|Animation|Children|Comedy|Fantasy
2809  Adventure|Animation|Children|Comedy|Fantasy
3000  Adventure|Animation|Children|Comedy|Fantasy
3568  Adventure|Animation|Children|Comedy|Fantasy
```

The function has generated a list of 5 movies that are most similar to the target movie (movie ID 1). The recommended movies share similar genres, which is evident from the output:

Recommended Movies:

1. **Antz (1998)**:
    - **Genres**: Adventure, Animation, Children, Comedy, Fantasy
2. **Toy Story 2 (1999)**:
    - **Genres**: Adventure, Animation, Children, Comedy, Fantasy
3. **The Adventures of Rocky and Bullwinkle (2000)**:
    - **Genres**: Adventure, Animation, Children, Comedy, Fantasy

4. **The Emperor's New Groove (2000)**:
    - **Genres**: Adventure, Animation, Children, Comedy, Fantasy
5. **Monsters, Inc. (2001)**:
    - **Genres**: Adventure, Animation, Children, Comedy, Fantasy

5. Hybrid Recommendations

The hybrid approach combines user-based, item-based, and content-based filtering to deliver more robust recommendations. This method leverages the strengths of each individual technique to address their weaknesses and improve overall accuracy.

```
# Hybrid Recommendation System
def hybrid_recommendations(user_id, movie_id, num_recommendations, user_item_matrix, user_similarity_df, item_similarity_df, movies, cosine_sim):
    user_based_recs = get_user_based_recommendations(user_id, num_recommendations, user_item_matrix, user_similarity_df)
    item_based_recs = get_item_based_recommendations(user_id, num_recommendations, user_item_matrix, item_similarity_df)
    content_based_recs = get_content_based_recommendations(movie_id, num_recommendations, movies, cosine_sim)

    user_based_movie_ids = [rec[0] for rec in user_based_recs]
    item_based_movie_ids = [rec[0] for rec in item_based_recs]
    content_based_movie_ids = content_based_recs['movieId'].tolist()

    combined_movie_ids = list(set(user_based_movie_ids + item_based_movie_ids + content_based_movie_ids))

    return combined_movie_ids

# Usage
hybrid_recs = hybrid_recommendations(user_id, movie_id, 5, user_item_matrix, user_similarity_df, item_similarity_df, movies, cosine_sim)
print("Hybrid Recommendations:", hybrid_recs)

[14]  ✓ 11.9s
```

Detailed Breakdown:

The recommendation system leverages three types of filtering techniques:

1. **User-Based Collaborative Filtering (UBCF)**:
    - Identifies users with similar tastes to the target user and recommends movies based on the preferences of these similar users.
    - Example Recommendation: **Movie ID 4993**
2. **Item-Based Collaborative Filtering (IBCF)**:
    - Recommends movies similar to those that the target user has rated highly.
    - Example Recommendation: **Movie ID 2918**
3. **Content-Based Filtering**:

- Recommends movies based on the attributes (genres, actors, directors, etc.) of a specific movie that the user likes.
- Example Recommendation: **Movie ID 3114**

```
...  Hybrid Recommendations: [4993, 2918, 2762, 3114, 1036, 589, 3754, 1968, 1200, 4016, 2294, 1527, 4886, 858, 318]
```

Combined Recommendations:

The combined list of recommendations ensures that the movies suggested are diverse and cater to different aspects of the user's preferences. Here are a few key examples from the output:

1. **Movie ID 4993**:
   - Likely recommended based on user-based collaborative filtering, indicating similar users enjoyed this movie.
2. **Movie ID 2918**:
   - Likely recommended based on item-based collaborative filtering, indicating this movie is similar to other movies the user has rated highly.
3. **Movie ID 3114**:
   - Likely recommended based on content-based filtering, indicating this movie shares attributes with a specific movie the user likes.
4. **Movie ID 1036**:
   - A result of combining recommendations from different filtering techniques, ensuring a broader appeal to the user's varied tastes.
5. **Movie ID 3754**:
   - Represents a fusion of user preferences and movie attributes, maximizing the relevance of the recommendation.

Practical Implications:

The hybrid recommendation system provides a well-rounded and balanced list of movie suggestions by leveraging multiple recommendation strategies. This approach enhances the recommendation accuracy, catering to various aspects of user preferences, and offers a diverse selection of movies for the user to explore. The hybrid recommendation system effectively combines user-based, item-based, and content-based filtering to deliver personalized and comprehensive movie recommendations. This ensures that the recommendations are highly relevant and tailored to the user's unique preferences, providing satisfying and engaging user experience.

6. Conclusion

This project demonstrates the implementation of a hybrid recommendation system that effectively integrates multiple recommendation techniques. The findings underscore the potential of hybrid approaches in enhancing user satisfaction by delivering accurate and diverse recommendations.

7. References

Papers and Algorithms

- **Ricci, F., Rokach, L., & Shapira, B. (2011). Introduction to Recommender Systems Handbook. Springer.**
    - This handbook provides a comprehensive overview of different recommendation systems, including collaborative filtering, content-based filtering, and hybrid methods.
- **Aggarwal, C. C. (2016). Recommender Systems: The Textbook. Springer.**
    - This textbook covers various recommender system algorithms and techniques, discussing their implementation and evaluation.
- **Salakhutdinov, R., & Mnih, A. (2008). Probabilistic Matrix Factorization. In Advances in Neural Information Processing Systems (pp. 1257-1264).**
    - This paper introduces matrix factorization techniques used in collaborative filtering.
- **Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent Dirichlet Allocation. Journal of Machine Learning Research, 3, 993-1022.**
    - This paper provides insights into content-based filtering using topic modeling approaches.
- **Scikit-learn documentation. (n.d.). Retrieved from https://scikit-learn.org/**
    - The documentation offers detailed explanations of the machine learning algorithms used, including TF-IDF and cosine similarity.