**Project Report**

**Movie Recommendation System using User-Based Collaborative Filtering**

**1. Introduction**

This project focuses on developing a fundamental Movie Recommendation System. The primary goal is to suggest movies to a user based on the preferences of similar users. This approach, known as **User-Based Collaborative Filtering**, is a classic technique in recommender systems, leveraging the idea that if two users share similar tastes in some movies, they are likely to have similar preferences for other movies.

The system is implemented in Python and utilizes the MovieLens ml-latest-small dataset, a widely recognized benchmark for recommendation systems.

**2. Dataset**

The dataset used for this project is the ml-latest-small dataset provided by MovieLens (GroupLens Research). It consists of two primary files:

* ratings.csv: Contains user ratings for movies, including userId, movieId, rating (on a 0.5 to 5.0 scale), and timestamp.
* movies.csv: Contains movie metadata, including movieId, title, and genres.

This dataset is ideal for building collaborative filtering models as it directly provides explicit user ratings, which are crucial for identifying user preferences and similarities. The dataset contains approximately 100,000 ratings from 610 users on 9,742 movies.

**3. Methodology**

The development of the Movie Recommendation System follows these key steps:

**3.1. Data Loading and Preparation**

The first step involves loading the ratings.csv and movies.csv files into Pandas DataFrames. These two DataFrames are then merged on the movieId column to combine rating information with movie titles. This consolidated DataFrame forms the basis for creating the user-item interaction matrix. The system also includes a utility to automatically download and extract the MovieLens dataset if it's not present locally, ensuring easy setup.

**3.2. Creating the User-Item Matrix**

A critical component for collaborative filtering is the **User-Item Matrix**. This matrix is created by pivoting the merged DataFrame, where:

* Rows represent unique userIds.
* Columns represent unique movie titles.
* Cell values are the rating given by a user to a specific movie.
* Movies that a user has not rated are represented by NaN (Not a Number) values. This sparse matrix efficiently captures user-movie interactions.

**3.3. Calculating User Similarity**

To identify users with similar tastes, **Cosine Similarity** is calculated between all pairs of users in the User-Item Matrix.

* **Cosine Similarity:** This metric measures the cosine of the angle between two user rating vectors. A value close to 1 indicates high similarity (users rated common movies similarly), while a value close to 0 suggests little to no similarity.
* **Handling Missing Values:** During similarity calculation, only movies that *both* users have rated are considered. This ensures that similarity is based solely on shared preferences, preventing NaN values from skewing the results.

The result is a symmetric **User-Similarity Matrix**, where each cell (i, j) represents the similarity score between User i and User j.

**3.4. Generating Recommendations**

For a given target\_user\_id, the system performs the following steps to recommend movies:

1. **Identify Unrated Movies:** It first determines which movies the target\_user\_id has not yet rated.
2. **Find Similar Users:** It retrieves the similarity scores of the target\_user\_id with all other users from the pre-computed User-Similarity Matrix. The users are then sorted by their similarity score in descending order.
3. **Predict Ratings:** For each unrated movie, the system predicts a rating by taking a **weighted average** of the ratings given by similar users who *have* rated that particular movie. The weights used in this average are the similarity scores between the target user and the similar users. Only similar users with a positive similarity score and who have rated the movie are considered. A minimum threshold (min\_common\_ratings) is also applied to ensure predictions are based on sufficient common ratings.
4. **Rank and Recommend:** The predicted ratings for all unrated movies are then sorted in descending order. The top num\_recommendations movies with the highest predicted ratings are presented as the final recommendations.

**4. Key Libraries Used**

* **Pandas:** Essential for data loading, manipulation, and creating the User-Item Matrix (DataFrame, read\_csv, merge, pivot\_table).
* **NumPy:** Used for numerical operations, especially in similarity calculations (np.array, np.linalg.norm, np.dot).
* **SciPy:** Specifically, scipy.spatial.distance.cosine is used for efficient calculation of cosine similarity between vectors.
* **os, zipfile, requests:** Used for programmatically downloading and extracting the MovieLens dataset, making the project self-contained.

**5. Strengths of User-Based Collaborative Filtering**

* **Intuitive:** The concept of "people who liked X also liked Y" is easy to understand.
* **No Item Metadata Needed:** It doesn't require any information about the movies themselves (genres, actors, etc.), only user ratings. This makes it versatile for various item types.
* **Discovers Unobvious Connections:** Can recommend items that are not directly related by genre or explicit tags but are liked by users with similar tastes.

**6. Limitations**

* **Scalability:** For very large numbers of users, calculating and storing the User-Similarity Matrix can be computationally expensive and memory-intensive.
* **Sparsity:** In many real-world scenarios, users rate only a small fraction of available movies. A highly sparse user-item matrix can lead to difficulty in finding truly similar users (the "cold user" problem).
* **Cold Start Problem:**
  + **New Users:** If a new user joins with no ratings, the system cannot find similar users or make recommendations for them.
  + **New Movies:** If a new movie is added with no ratings, it won't be recommended until it gathers sufficient ratings from users.
* **Performance:** Simple averaging might not capture complex rating patterns effectively.

**7. Future Enhancements**

To improve this recommendation system, several enhancements can be explored:

* **Item-Based Collaborative Filtering:** Implement an item-based approach (where similarity between movies is calculated) which often scales better for a large number of users.
* **Matrix Factorization Techniques:** Explore more advanced model-based methods like Singular Value Decomposition (SVD) or Alternating Least Squares (ALS) using libraries like Surprise. These can discover latent factors and handle sparsity more effectively.
* **Content-Based Filtering:** Integrate movie metadata (genres, plot summaries, keywords) to recommend movies based on their inherent characteristics. This helps address the cold-start problem for new movies.
* **Hybrid Recommendation Systems:** Combine collaborative filtering with content-based filtering to leverage the strengths of both approaches.
* **Evaluation Metrics:** Implement robust evaluation metrics (e.g., RMSE, Precision, Recall, F1-score) and cross-validation techniques (e.g., TimeSeriesSplit or KFold for recommendation systems) to systematically assess model performance.
* **Larger Datasets:** Experiment with larger MovieLens datasets (e.g., 20M or 25M) to test scalability and performance.
* **User Interface:** Develop a simple web interface (e.g., using Flask or Streamlit) to interact with the recommendation system more easily.

**8. Conclusion**

This project successfully demonstrates the implementation of a User-Based Collaborative Filtering movie recommendation system. By leveraging user rating patterns and cosine similarity, it can suggest personalized movie recommendations. While this serves as a strong foundational understanding, the noted limitations highlight areas for further exploration and the application of more advanced techniques to build a truly robust and scalable recommendation engine.