Introduction

In today's digital age, users of social media and video streaming services are bombarded with an overwhelming array of choices, often resulting in information overload. This makes selecting a movie or content a challenging and time-consuming task.

My goal is to address this problem by developing a recommendation system based on user ratings. Specifically, I aim to create a movie recommendation system that analyzes a user's past rating behavior to understand their preferences and predict the scores of both watched and unwatched movies. This system will then automatically suggest content that the user is likely to enjoy.

The impact of solving this problem is far-reaching. Video streaming services like YouTube, Netflix, and Paramount have revolutionized our society, transforming our habits and lifestyles. Moreover, these service providers with successful recommendation systems generate substantial profits by enhancing user satisfaction and engagement.

Formulation:

This problem is formulated as a recommendation task, and I implemented it using Collaborative Filtering and Cosine Similarity. The objective is to predict a user's rating for an item they haven't rated yet.

The most crucial input in my dataset is a set of historical user-item interactions extracted from the ratings.csv file. Each row in this data represents a userID, movieID, and rating. The preprocessing step involves transforming this data into a large, sparse user-item matrix. The left side of the matrix contains a column of all userIDs, while the top side contains a row of all movieIDs. The matrix is sparse because not every user has watched every movie; most users only watch a small subset of all movies. Consequently, the matrix will be filled with NaN values.

The expected output is a predicted rating for every (userID, movieID) input pair, ranging from 0.0 to 5.0. These predictions can then be sorted to generate a Top-k list of movie recommendations for a specific user.

Dataset:

I obtained my dataset from Kaggle, specifically the "Movie Lens Small Latest Dataset." This dataset is a subset of the MovieLens dataset, which is maintained by the GroupLens research lab at the University of Minnesota.

The Movie Lens Small Latest Dataset comprises 610 users, and each user has rated at least 20 movies. Notably, no demographic information is included in the dataset. Each user is represented by a unique ID, and no additional information is provided. The dataset contains 9,724 movies, with the largest movie ID being 193,587. This suggests that the user-item matrix is expected to be sparse and wide. The ratings are provided on a 5-star scale, with half-star increments.

Data preprocessing involves loading the data, dropping unused columns, and performing data transformations. The loading process involves loading the ratings.csv file into Pandas data frames. Next, I removed the timestamp column, as it was not required. Finally, the most crucial step is to pivot the data frames to create the user-item matrix. As mentioned earlier, the matrix is highly sparse, with most entries being NaN. Since the cosine similarity algorithm cannot handle NaN values, I calculate the mean of the non-NaN values in each row. Then, I subtract this mean from the non-NaN values in that row and fill the NaN values with 0. This process effectively sets the average rating for each user to 0.

Algorithm:

The data mining algorithm I applied is user-based collaborative filtering (UBCF). The reason for choosing this algorithm is that it is a memory-based, non-model approach perfectly suited for solving the recommendation task based on the rating data. UBCF with metrics like cosine similarity is simple and it is effective with spare data. In addition, UBCF has a strong performance baseline. UBCF provides a robust baseline performance that is often competitive with more complex model-based methods.

The algorithm starts by constructing a user similarity matrix, which maps out the similarity between all users. This matrix is created by calculating the similarity between every pair of

users based on their shared rating history in the training data. In this project, I used Cosine Similarity, a widely used method for determining this likeness. Cosine Similarity measures the cosine of the angle between two users' ratings, resulting in a score ranging from -1 to 1. A score close to 1 indicates that the users have very similar rating patterns, while a score close to 0 suggests a lack of relationship. For instance, the diagonal of the similarity matrix should be 1. Once the similarity matrix is established, the algorithm predicts a rating for an unrated movie by identifying the Top K most similar users and calculating a weighted average of the ratings those neighbors gave to the target movie. The similarity scores serve as the weights for this calculation.

Experiments:

To illustrate the algorithm's functionality, let's delve into the case study first. In this instance, I'll generate a list of the top 10 most similar users to User with ID 1 and predict the score for a movie with ID 1. The top 10 most similar users are listed below:

Neighbor ID	Similarity Score to User 1	Rating on the Movie with ID 1
301	0.125	nan
597	0.103	4.0
414	0.101	4.0
477	0.099	4.0
57	0.098	5.0
369	0.097	nan
206	0.097	5.0
535	0.096	nan
590	0.095	4.0
418	0.094	nan

After identifying the top ten most similar users to User 1 and obtaining their ratings for Movie with ID 1, I calculated the predicted rating using a weighted average. Only users who had actually rated the movie were considered, disregarding any NaN values. In this calculation, each neighbor's rating was weighted by their similarity score. The final prediction was determined by dividing the sum of the products of similarity scores and ratings by the sum of the similarity scores used in the calculation.

For this case study, the predicted rating for User 1 on Movie ID1 is 4.33, while the actual rating in the training dataset is 4.0. This approach gives more weight to users who are more similar to User 1, resulting in a more personalized and accurate prediction.

The primary evaluation metric used is the root mean squared error (RMSE) as it is ideal for measuring the magnitude of the error for my prediction. The reason that I choose RMSE instead of mean absolute error (MAE) is that it heavily penalizes large errors, which is useful in cases where a large deviation is undesirable. For example, predicting 0.5 star while use gave a 5.0 star rating.

The formula of RMSE is:

$$RMSE = \sqrt{\left(\frac{1}{n}\right)\sum(\hat{y}_i - y_i)^2}$$

Where n = number of observations, $\hat{y}_i$ = predicted value, and y_i = actual value.

My prediction algorithm's RMSE is 0.97, which is lower than the baseline RMSE of 1.05 for the MovieLens 100k dataset and 0.93 for the MovieLens 1M dataset. This indicates that my algorithm and data preprocessing are meaningful and yield promising results.

Optional:

Advanced analytics: What challenges do you find in the data? How do you tackle these challenges?

The challenge I encountered in the data was the sparsity of the user-item matrix. Since each user only rated a limited selection of movies, the matrix was filled with missing values (NaN). Therefore, the method used to replace NaN was crucial to the performance of the recommendation system.

Initially, I set all NaN values to 0 and didn't make any changes to the other non-NaN ratings. However, after running the first iteration, the RMSE value was 1.92, which was far from the baseline performance. This was because this approach caused distortions in the rating patterns and biased the user preferences.

To address this issue, I adopted the most commonly used and intuitive method, the mean-centering approach. After applying this approach to the Dataset section (as mentioned in the third paragraph), the performance significantly improved, and the RMSE dropped substantially. This demonstrated the critical role of data preprocessing in data mining tasks, especially with collaborative filtering systems that deal with sparse data.

Another challenge I faced was when I was trying to find the Top K most similar users for a user and make a specific movie recommendation. Sometimes, all the top K similar users hadn't seen the target movie yet, so I returned 0 as the predicted rating. This also resulted in poor performance. Fortunately, addressing this issue was straightforward. I simply wrote a conditional function to check if any of the top K similar users had watched the movie. If none of them had watched it, instead of returning 0, I returned the average rating of the entire dataset. This fallback strategy prevented invalid predictions and ensured that the prediction remained neutral even when no similar users had seen the movie.

With these two improvements—mean-centering and handling unrated movies during Top-K prediction—the RMSE decreased dramatically from 1.92 to 0.97, demonstrating the effectiveness of the preprocessing and prediction refinements.

AI usage disclosure statement:

Did you use any AI tools (such as ChatGPT, MicrosoftCoPilot, or similar) in completing this project? If so, please briefly describe how you used them (e.g., writing the codes, writing the report, etc.) and which specific AI tools you used. If not, simply answer 'No'.

Yes, I did. I used Apple's built-in AI writing tool to rewrite my report because my English proficiency is limited. Additionally, I asked ChatGPT to help me make some of my sentences more fluent. However, I did not use any AI tools to generate ideas.

Regarding the code, I used Copilot to generate comments.