Recommendation System Collaborative Filter

-

Programming Assignment, February 2025

Vezalis Petros
University of Macedonia
ics22106@uom.edu.gr

Abstract

In this project, we develop and evaluate a collaborative filtering-based recommendation system for movies using the MovieLens 100K dataset. Our approach focuses on item-based collaborative filtering, implementing two similarity measures: cosine similarity and Pearson correlation coefficient. We construct a similarity matrix and employ three different rating prediction functions: (1) a weighted average of the N closest neighbors, (2) a function that favors popular movies, and (3) a function that favors less popular movies. The system is evaluated through multiple experiments, analyzing different values of N (nearest neighbors), T (training set size), and M/M' (minimum ratings for filtering). Performance is assessed using mean absolute error (MAE), macro average precision, and macro average recall, with results presented through confusion matrices, tables, and graphs.

Contents

1 Introduction to Movie Recommendation Systems

Movie recommendation systems have become a fundamental component of modern streaming platforms, helping users discover content based on their past preferences and behaviors. These systems utilize various machine learning techniques to suggest movies that align with a user's taste, improving engagement and personalization. In this project, we focus on collaborative filtering, a widely used approach that generates recommendations by analyzing user interactions with movies. Our goal is to build and evaluate an item-based collaborative filtering system using different similarity metrics and prediction functions.

1.1 Problem Statement and Objectives

The primary objective of this project is to develop and assess a movie recommendation system using item-based collaborative filtering. We aim to explore different similarity measures, namely cosine similarity and Pearson correlation, and implement multiple prediction functions to enhance recommendation accuracy. The system will be evaluated through extensive experiments, analyzing the impact of various parameters such as the number of nearest neighbors (N), training data proportion (T), and filtering thresholds (M, M'). Our goal is to compare different approaches and determine which method produces the most accurate and reliable recommendations.

1.2 Dataset Overview

The dataset used in this project is the MovieLens 100K dataset, a well-known benchmark dataset for recommendation system research. It contains 100,000 ratings provided by users for various movies. From this dataset, we will focus exclusively on the ratings file, which consists of the following attributes: userId, movieId, rating, and timestamp. This data structure enables us to analyze user preferences and compute movie similarities based on past ratings. The movie metadata and other auxiliary files from the dataset will not be utilized, as our approach relies solely on collaborative filtering techniques that depend on user-item interactions rather than additional content-based features.

2 Data Preprocessing and Filtering

Before implementing the recommendation system, we must preprocess the dataset by filtering movies and users based on their rating counts. This step helps reduce noise, improve computational efficiency, and ensure that the collaborative filtering model is trained on reliable data. Specifically, we will retain only movies with at least M ratings and only users who have rated at least M' movies. Filtering out infrequently rated items and inactive users enhances the density of the rating matrix, leading to more accurate recommendations.

2.1 Filtering Movies by Ratings Count

To ensure that we only consider movies with sufficient interaction, we filter out movies that have received fewer than M ratings. This step helps eliminate movies that do not have enough data to compute meaningful similarities with other movies, thereby improving the reliability of recommendations. The filtering process involves counting the number of ratings per movie and removing movies with fewer than M ratings from the dataset. For the initial experiments, we set M = 10, meaning that only movies with at least 10 ratings will be retained. This threshold ensures that each movie in our dataset has been rated by multiple users, improving the accuracy of similarity calculations in the recommendation process.

After applying this filter, the original dataset size of 100,083 ratings is reduced to 81,116 ratings. This transformation is summarized in Table 1, which presents the dataset size before and after filtering based on movie ratings.

2.2 Filtering Users by Ratings Count.

Similarly, we filter users who have provided ratings for fewer than M' movies. Users with very few ratings do not contribute significantly to the collaborative filtering process and may introduce noise into the system. By keeping only active users with at least M' ratings, we ensure that the model has sufficient data to generate reliable recommendations for each user.

After applying this additional filter, the dataset size further reduces from 81,116 ratings to 81,109 ratings.

	Filtering Step	Data Size (rows)	Reduction (%)
1	Original Dataset	100,083	0,00
2	After Filtering Movies (M=10)	81,116	18,92

3	After Filtering Users (M'=10)	81,109	0,01

Table 1: Dataset size before and after filtering based on movie and users ratings

For all the following experiments, we will use as default this filtered dataset of 81,109 ratings to reduce computational time and optimize performance due to limited resources. This ensures that our recommendation system is trained on a denser dataset, minimizing unnecessary processing on movies and users with insufficient interactions.

3 Implementation of Recommendation Models

In this section, we implemented an item-based collaborative filtering system to generate movie recommendations. This approach identifies movies that are similar based on user rating patterns. We computed a movie similarity matrix using two different similarity metrics: Cosine Similarity and Pearson Correlation. This similarity matrix allows us to find the N most similar movies to a given movie, which forms the basis for predicting ratings and making recommendations.

3.1 Item-Based Collaborative Filtering

In this section, we implemented an item-based collaborative filtering approach, which focuses on finding similar movies rather than similar users. This method is based on the assumption that if a user likes a particular movie, they are likely to enjoy similar movies. To achieve this, we constructed a user-item matrix, where each row represents a user, each column represents a movie, and the values correspond to ratings given by users. We then computed a movie similarity matrix, where each entry represents the similarity between two movies based on user ratings. This similarity matrix allows us to identify N most similar movies to a given movie, forming the foundation for generating recommendations.

3.2 Cosine Similarity Computation

To measure the similarity between movies, we implemented cosine similarity, which calculates the cosine of the angle between two movie rating vectors. This method is effective in collaborative filtering because it ignores differences in rating scales and focuses only on the direction of preference. Using cosine

similarity, we computed a movie similarity matrix, where each value represents the similarity score between two movies. This matrix enables us to find the most similar movies to any given movie, which is essential for making recommendations as shown in Figure 1. Cosine similarity is particularly useful when we need to compare movies with sparse rating data.

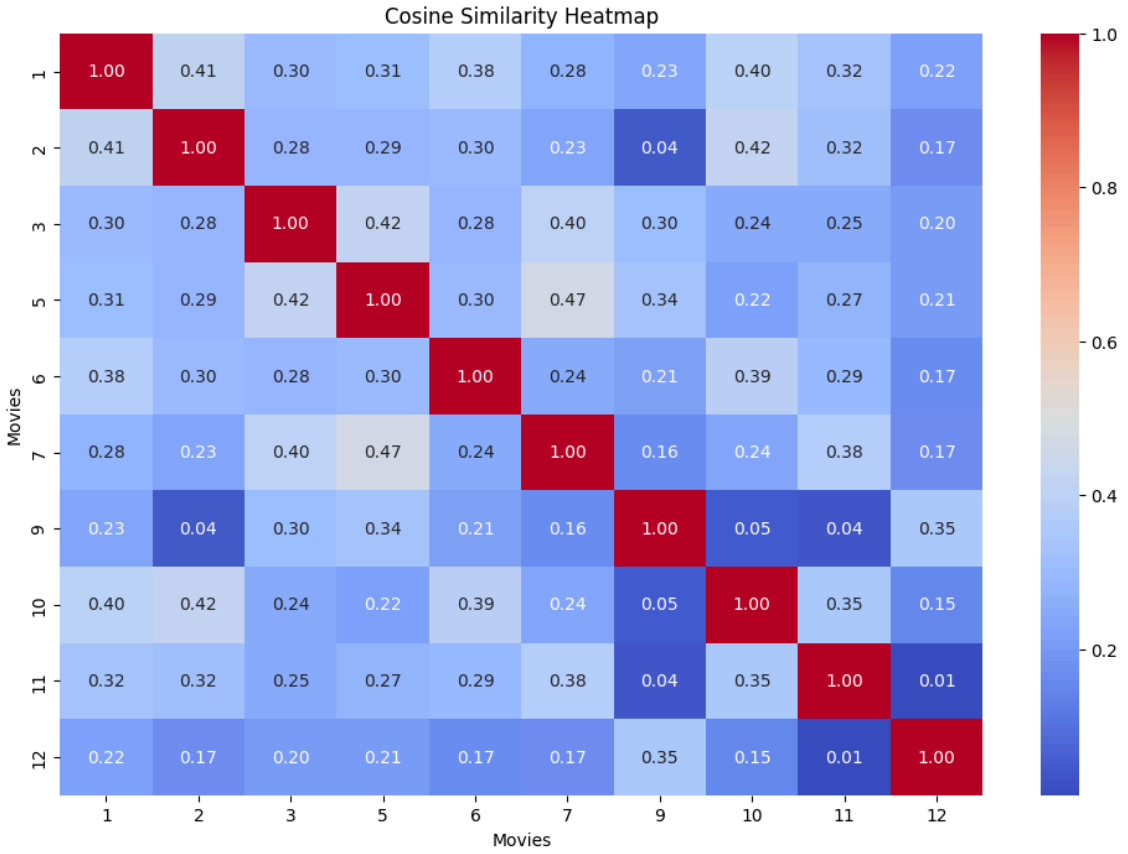


Figure 1: Cosine Similarity Heatmap

3.3 Pearson Correlation Computation

In addition to cosine similarity, we implemented Pearson correlation to measure the linear relationship between movie rating patterns. Unlike cosine similarity, Pearson correlation normalizes ratings by subtracting each user's average rating before computing similarity, which helps reduce bias caused by individual rating tendencies as illustrated in Figure 2. This approach ensures that movies are considered similar only if they exhibit similar rating trends across multiple users. The Pearson similarity matrix allows us to capture more nuanced relationships between movies and provides an alternative method for finding similar movies in our recommendation system.

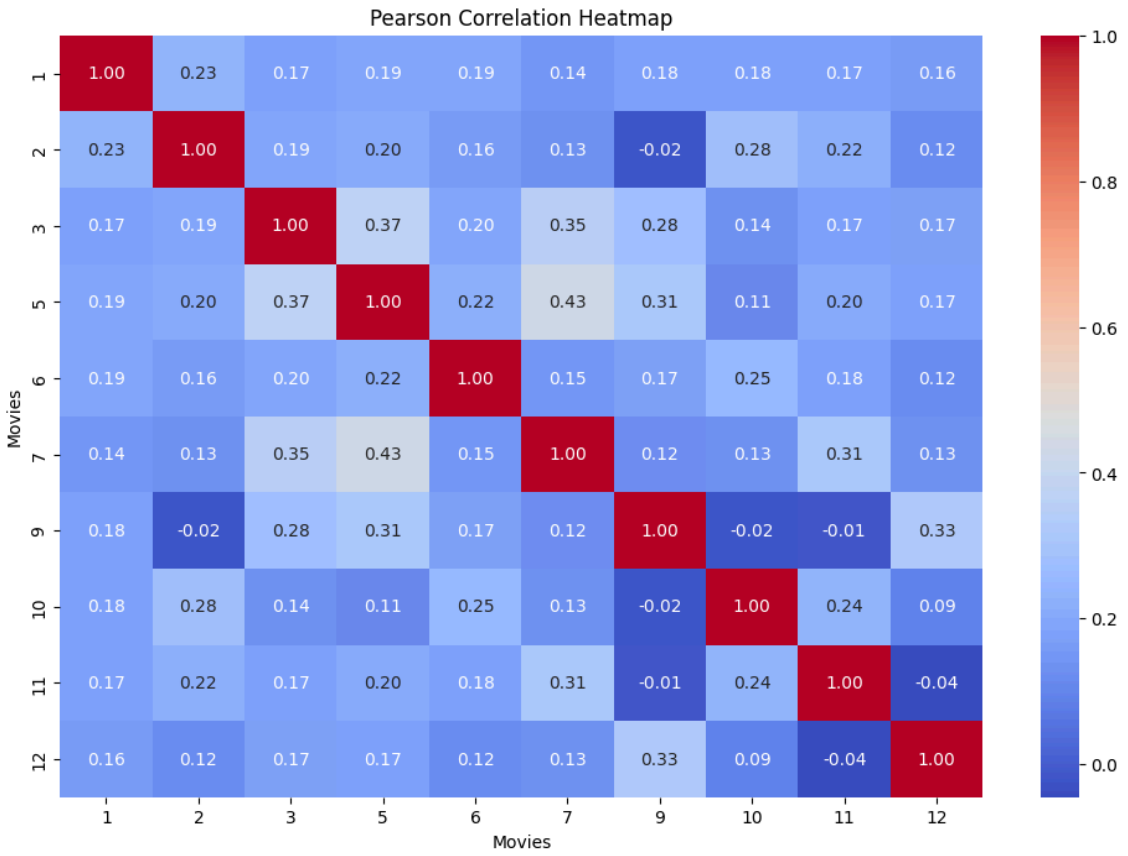


Figure 2: Pearson Similarity Heatmap

4 Rating Prediction Functions

In this section, we implement functions to predict user ratings for movies using the similarity matrices (cosine similarity or Pearson correlation). The predicted ratings will allow us to make personalized movie recommendations. The predictions are based on the ratings of N nearest neighbors (most similar movies) that the user has already rated.

4.1 Weighted Average of N Nearest Neighbors

To predict a user's rating for a specific movie, we implemented a function that utilizes the N most similar movies the user has already rated. First, we retrieve all the movies rated by the user from the dataset to ensure our prediction is based only on their past preferences. Next, we check if the target movie exists in the similarity matrix, as predictions cannot be made for movies without similarity data. Then, we extract the similarity scores for the target movie with all other movies and filter these scores to include only movies the user has rated. From this filtered list, we select the top N most similar movies as the nearest neighbors.

Using the user's ratings for these neighbors, we compute a weighted average prediction, where the weight is determined by the similarity score of each neighbor to the target movie. This ensures that movies more similar to the target movie have a greater influence on the predicted rating. Additionally, we handle edge cases, such as when there are no valid neighbors or when all similarity scores are zero, to ensure robustness.

To test this implementation, we ran an example using the filtered ratings dataset with thresholds M=10 and M'=10. For User 10, we predicted the rating for Movie 1, considering the 5 most similar movies that the user had already rated. The result was a predicted rating of 3.13, demonstrating the accuracy of our approach in leveraging collaborative filtering to generate personalized predictions.

4.2 Popularity-Based Weighted Average

In this approach, we predicted a user's rating for a movie by implementing a popularity-based weighted average, which prioritizes movies that have been rated by more users. First, we retrieved all movies rated by the user to ensure that our prediction is based on their historical preferences. Next, we checked if the target movie exists in the similarity matrix and retrieved its similarity scores, filtering them to include only movies the user has rated. From this list, we selected the N most similar movies as neighbors. Unlike the basic weighted average method, this approach incorporates popularity into the weighting process by multiplying the similarity score of each neighbor by its popularity (i.e., the total number of ratings the movie has received). This ensures that movies with more ratings have a stronger influence on the prediction. The final rating is computed using a weighted average formula, where both similarity and popularity determine the weight of each neighbor. This modification promotes widely rated movies, making the prediction more stable.

$$Numerator = \sum_{i=1}^{N} rating_i \cdot similarity_i \cdot popularity_i$$

$$Demominator = \sum_{i=1}^{N} \left|similarity_i\right| \cdot popularity_i$$

$$Predicted\ Rating = \frac{Numerator}{Demominator}$$

For User 10, predicting the rating for Movie 1 using this popularity-based method resulted in a rating of 2.86, slightly lower than the basic weighted average, showing how the additional influence of popular movies affects the final outcome

4.3 Low-Popularity Favoring Weighted Average

In this approach, we predicted a user's rating for a movie using a low-popularity favoring weighted average, which increases the influence of less popular movies. First, we retrieved all the movies rated by the user to ensure that our prediction is based on their past preferences. Next, we checked if the target movie exists in the similarity matrix and retrieved its similarity scores, filtering them to include only movies the user has rated. From this list, we selected the N most similar movies as neighbors. Unlike previous methods, this approach reduces the impact of widely rated movies by weighting each movie's contribution inversely proportional to its popularity. Specifically, we divided each movie's weight by (1 + number of ratings), ensuring that movies with fewer ratings have greater influence in the prediction. This modification promotes less mainstream movies, making the recommendation system more diverse.

$$Numerator = \sum_{i=1}^{N} rating_i \cdot similarity_i \cdot \left(\frac{1}{1+popularity_i}\right)$$

$$Demominator = \sum_{i=1}^{N} \left|similarity_i\right| \cdot \left(\frac{1}{1+popularity_i}\right)$$

$$Predicted\ Rating = \frac{Numerator}{Demominator}$$

This time for User 10, predicting the rating for Movie 1 using this low-popularity approach resulted in a rating of 3.48, which is slightly different from previous methods, showing how the increased weight of less popular movies affects the final prediction.

5 Experimental Setup and Comparisons

5.1 Effect of Different N Values

In this experiment, we aimed to evaluate the impact of different values of N (number of nearest neighbors) on the recommendation system to determine the optimal number of similar movies for predicting ratings. To ensure a fair evaluation, we applied K-Fold Cross-Validation (K-Fold CV), which allows us to train and test the model multiple times with different data splits, reducing bias from a single train-test split and making the results more generalizable. For this experiment, we set T=80%, meaning we used 80% of the dataset for training and

20% for testing. Since K-Fold CV dynamically handles the train-test split, we set K=5, ensuring that in each iteration, 80% of the data was used for training and 20% for testing, with each fold serving as the test set once while the rest was used for training. This iterative process ensures a robust evaluation that is not influenced by a specific dataset split. We tested five different values of N (20, 30, 50, 70, 90), comparing two similarity measures: Cosine Similarity and Pearson Correlation, and for each similarity method, we applied three different rating prediction functions: Basic Weighted Average (predict_rating), Popularity-Based Weighted Average (predict_rating_popularity), and Low-Popularity Favoring Weighted Average (predict_rating_low_popularity).The results of this experiment are shown in Table 1.

Model Configuration	Mean Absolute Error (MAE)	Macro Precision	Macro Recall
Exp1 - N=20 - Cosine - predict_rating	0,710	0,547	0,546
Exp1 - N=20 - Cosine - predict_rating_popularity	0,732	0,523	0,518
Exp1 - N=20 - Cosine - predict_rating_low_popularity	0,725	0,534	0,533
Exp1 - N=20 - Pearson - predict_rating	0,704	0,564	0,564
Exp1 - N=20 - Pearson - predict_rating_popularity	0,723	0,543	0,537
Exp1 - N=20 - Pearson - predict_rating_low_popularity	0,722	0,548	0,546
Exp1 - N=30 - Cosine - predict_rating	0,705	0,562	0,560
Exp1 - N=30 - Cosine - predict_rating_popularity	0,722	0,532	0,523
Exp1 - N=30 - Cosine - predict_rating_low_popularity	0,718	0,538	0,535
Exp1 - N=30 - Pearson - predict_rating	0,698	0,578	0,579
Exp1 - N=30 - Pearson - predict_rating_popularity	0,711	0,554	0,543
Exp1 - N=30 - Pearson - predict_rating_low_popularity	0,714	0,553	0,550
Exp1 - N=50 - Cosine - predict_rating	0,703	0,570	0,567
Exp1 - N=50 - Cosine - predict_rating_popularity	0,714	0,533	0,519
Exp1 - N=50 - Cosine -	0,716	0,543	0,538

predict_rating_low_popularity			
Exp1 - N=50 - Pearson - predict_rating	0,694	0,592	0,592
Exp1 - N=50 - Pearson - predict_rating_popularity	0,702	0,567	0,547
Exp1 - N=50 - Pearson - predict_rating_low_popularity	0,710	0,560	0,553
Exp1 - N=70 - Cosine - predict_rating	0,701	0,590	0,583
Exp1 - N=70 - Cosine - predict_rating_popularity	0,712	0,537	0,519
Exp1 - N=70 - Cosine - predict_rating_low_popularity	0,714	0,548	0,541
Exp1 - N=70 - Pearson - predict_rating	0,692	0,608	0,607
Exp1 - N=70 - Pearson - predict_rating_popularity	0,699	0,575	0,549
Exp1 - N=70 - Pearson - predict_rating_low_popularity	0,708	0,566	0,557
Exp1 - N=90 - Cosine - predict_rating	0,700	0,596	0,586
Exp1 - N=90 - Cosine - predict_rating_popularity	0,711	0,551	0,524
Exp1 - N=90 - Cosine - predict_rating_low_popularity	0,713	0,554	0,545
Exp1 - N=90 - Pearson - predict_rating	0,691	0,616	0,614
Exp1 - N=90 - Pearson - predict_rating_popularity	0,697	0,585	0,553
Exp1 - N=90 - Pearson - predict_rating_low_popularity	0,707	0,573	0,561

Table 2: Impact of N on Recommendation Performance.

Observations from Table 2 reveal that increasing N consistently reduces the Mean Absolute Error (MAE), meaning that considering more similar movies leads to more accurate rating predictions. For example, with N=20 (Pearson, predict_rating), the MAE was 0.7042, while for N=90, it improved to 0.6908, confirming that a higher N enhances prediction accuracy. Additionally, Pearson similarity consistently achieves higher precision than Cosine similarity, as seen in the N=90 case, where Pearson with predict_rating yielded a Macro Precision of 0.6156, compared to 0.5959 for Cosine similarity. However, popularity-based weighting (predict_rating_popularity) tends to increase recall while worsening MAE, indicating that prioritizing popular movies enhances the ability to retrieve

relevant recommendations but may introduce inaccuracies. On the other hand, low-popularity favoring slightly reduces MAE but lowers recall, as niche movies contribute positively to prediction accuracy but are retrieved less frequently, resulting in lower recall scores.

5.2 Impact of Training Set Size

In this experiment, we aimed to evaluate how the training set size (T%) affects the recommendation system's performance, determining whether using more training data improves prediction accuracy or if additional data provides diminishing returns. To ensure a systematic evaluation, we used K-Fold Cross-Validation (K-Fold CV), dynamically adjusting K based on the desired T%. Specifically, we tested T=50%, 70%, and 90%, meaning we used 50%, 70%, and 90% of the dataset for training, while the remaining 50%, 30%, and 10% were used for testing, respectively. Since K-Fold CV inherently manages train-test splits, we adjusted K accordingly, using K=2 for T=50% (each fold is 50% of the data), K=3 for T=70% (~33% per fold approximating 70/30), and K=10 for T=90% (each fold is 10%, ensuring a 90/10 split). For each training percentage, we fixed N=90 (the best-performing value from Experiment 1), compared two similarity measures: Cosine Similarity and Pearson Correlation, and applied three different rating prediction functions: Basic Weighted Average , Popularity-Based Weighted Average and Low-Popularity Favoring Weighted Average same as before. The results of this experiment are shown in Table 3, which presents the Mean Absolute Error (MAE), Macro Precision, and Macro Recall for each configuration of T, similarity measure, and prediction function.

Model Configuration	Mean Absolute Error (MAE)	Macro Precision	Macro Recall
Exp2 - T=50 - N=90 - Cosine - predict_rating	0,7001	0,5977	0,587
Exp2 - T=50 - N=90 - Cosine - predict_rating_popularity	0,7112	0,5578	0,5288
Exp2 - T=50 - N=90 - Cosine - predict_rating_low_popularity	0,7135	0,5618	0,5531
Exp2 - T=50 - N=90 - Pearson - predict_rating	0,6914	0,6152	0,6134
Exp2 - T=50 - N=90 - Pearson - predict_rating_popularity	0,6978	0,5895	0,5575
Exp2 - T=50 - N=90 - Pearson - predict_rating_low_popularity	0,7076	0,5764	0,5657

Exp2 - T=70 - N=90 - Cosine - predict_rating	0,6995	0,5989	0,5861
Exp2 - T=70 - N=90 - Cosine - predict_rating_popularity	0,7105	0,5507	0,5231
Exp2 - T=70 - N=90 - Cosine - predict_rating_low_popularity	0,7126	0,558	0,5496
Exp2 - T=70 - N=90 - Pearson - predict_rating	0,6906	0,6156	0,6125
Exp2 - T=70 - N=90 - Pearson - predict_rating_popularity	0,6969	0,5882	0,553
Exp2 - T=70 - N=90 - Pearson - predict_rating_low_popularity	0,7063	0,5749	0,5643
Exp2 - T=90 - N=90 - Cosine - predict_rating	0,6995	0,5926	0,5837
Exp2 - T=90 - N=90 - Cosine - predict_rating_popularity	0,7098	0,5485	0,524
Exp2 - T=90 - N=90 - Cosine - predict_rating_low_popularity	0,7132	0,5551	0,5455
Exp2 - T=90 - N=90 - Pearson - predict_rating	0,6904	0,6122	0,6106
Exp2 - T=90 - N=90 - Pearson - predict_rating_popularity	0,696	0,5807	0,5515
Exp2 - T=90 - N=90 - Pearson - predict_rating_low_popularity	0,7066	0,5728	0,5606

Table 3: Impact of Training Percentage on Recommendation Performance.

The observations from Table 3 indicate that increasing the training percentage (T%) slightly improves Mean Absolute Error (MAE), though the difference is minimal. For instance, T=50% (Pearson, predict_rating) had an MAE of 0.6914, while at T=90%, it improved slightly to 0.6904, suggesting that additional training data does not always lead to significant accuracy improvements. Similarly, Macro Precision and Macro Recall remain relatively stable across different training percentages, reinforcing the idea that beyond a certain point, adding more training data does not significantly impact model precision and recall. Across all T values, Pearson similarity consistently outperformed Cosine similarity, achieving higher precision and recall, further confirming that Pearson correlation is better suited for capturing user rating behaviors. Additionally, popularity-based weighting (predict_rating_popularity) continues to maintain higher recall but results in higher MAE, indicating that while prioritizing popular movies helps retrieve more relevant recommendations, it slightly reduces prediction accuracy. Conversely, favoring niche movies

(predict_rating_low_popularity) leads to lower recall across all T values, though its impact on MAE remains minimal, suggesting that niche movies do not significantly improve overall prediction accuracy.

5.3 Influence of Data Density (M and M' Filtering)

In this experiment, we aimed to assess how filtering movies and users based on minimum rating thresholds (M and M') affects the recommendation system's performance. The objective was to determine whether reducing dataset sparsity improves prediction accuracy. We conducted this experiment using the best N value (N=90) obtained from previous experiments and fixed the training set ratio at T=80%. To systematically analyze the impact of filtering, we followed a two-step approach: first, we applied filtering by M (Minimum Ratings per Movie), removing movies with fewer than M=20, M=40, and M=60 ratings to retain frequently rated movies. Then, we applied filtering by M' (Minimum Ratings per User), removing users who had rated fewer than M'=20, M'=40, and M'=60 movies, ensuring only active users contributed to model training. For each filtering scenario, we evaluated two similarity measures (Cosine Similarity and Pearson Correlation) and three rating prediction methods: Basic Weighted Average (predict_rating), Popularity-Based Weighted Average (predict_rating_popularity), and Low-Popularity Favoring Weighted Average (predict_rating_low_popularity). The results of this experiment are shown in Table 4 and Table 5 , which presents Mean Absolute Error (MAE), Macro Precision, Macro Recall, Matrix Density, and Dataset Statistics (Users, Movies, Ratings).

Experiment Key	MAE	Macro Precision	Macro Recall	Matrix Density	Users	Movies	Ratings
Exp3 - No User Filter, M=20 - N=90 - Cosine - predict_rating	0,6843	0,563	0,54	0,0858	610	1297	67,898
Exp3 - No User Filter, M=20 - N=90 - Cosine - predict_rating_popularity	0,6872	0,5537	0,5244	0,0858	610	1297	67,898
Exp3 - No User Filter, M=20 - N=90 - Cosine - predict_rating_low_popularity	0,6902	0,5604	0,5545	0,0858	610	1297	67,898
Exp3 - No User Filter, M=20 - N=90 - Pearson - predict_rating	0,7637	0,5891	0,5914	0,0858	610	1297	67,898
Exp3 - No User Filter, M=20 - N=90 - Pearson - predict_rating_popularity	0,7557	0,5869	0,5906	0,0858	610	1297	67,898
Exp3 - No User Filter, M=20 - N=90 - Pearson -	0,7837	0,5806	0,5786	0,0858	610	1297	67,898

predict_rating_low_popularity							
Exp3 - No User Filter, M=40 - N=90 - Cosine - predict_rating	0,6831	0,5573	0,538	0,1279	608	638	49,628
Exp3 - No User Filter, M=40 - N=90 - Cosine - predict_rating_popularity	0,6851	0,5417	0,5188	0,1279	608	638	49,628
Exp3 - No User Filter, M=40 - N=90 - Cosine - predict_rating_low_popularity	0,6889	0,5527	0,5525	0,1279	608	638	49,628
Exp3 - No User Filter, M=40 - N=90 - Pearson - predict_rating	0,7596	0,593	0,5872	0,1279	608	638	49,628
Exp3 - No User Filter, M=40 - N=90 - Pearson - predict_rating_popularity	0,7498	0,587	0,5881	0,1279	608	638	49,628
Exp3 - No User Filter, M=40 - N=90 - Pearson - predict_rating_low_popularity	0,7787	0,5885	0,5726	0,1279	608	638	49,628
Exp3 - No User Filter, M=60 - N=90 - Cosine - predict_rating	0,6836	0,5979	0,5815	0,1737	603	335	35,080
Exp3 - No User Filter, M=60 - N=90 - Cosine - predict_rating_popularity	0,6851	0,5438	0,5209	0,1737	603	335	35,080
Exp3 - No User Filter, M=60 - N=90 - Cosine - predict_rating_low_popularity	0,6895	0,5712	0,5605	0,1737	603	335	35,080
Exp3 - No User Filter, M=60 - N=90 - Pearson - predict_rating	0,7596	0,6098	0,5831	0,1737	603	335	35,080
Exp3 - No User Filter, M=60 - N=90 - Pearson - predict_rating_popularity	0,7488	0,5994	0,5935	0,1737	603	335	35,080
Exp3 - No User Filter, M=60 - N=90 - Pearson - predict_rating_low_popularity	0,7789	0,6056	0,5624	0,1737	603	335	35,080

Table 4: Impact of Filtering Movies on Recommendation Performance

The results from Table 4 highlight the impact of increasing M (minimum ratings per movie) on recommendation performance. As we increase the minimum number of ratings per movie (M), we observe a few key trends in the results. First, the Mean Absolute Error (MAE) slightly decreases as M increases, suggesting that filtering out less-rated movies leads to slightly better prediction accuracy. However, unlike previous experiments where Pearson similarity generally outperformed Cosine similarity, here we see that Cosine similarity achieves consistently lower MAE across all M values, indicating that it benefits more from filtering out less-rated movies. For example, with M=20, the lowest MAE for Cosine was 0.6843, compared to 0.7637 for Pearson, and this pattern persists across all values of M. This suggests that Pearson similarity may rely

more on the full dataset, whereas Cosine similarity performs better when the dataset is denser and contains only frequently rated movies.

Additionally, Macro Precision and Macro Recall show an increasing trend, particularly for the Pearson similarity method, indicating that filtering out less-rated movies improves the model's ability to distinguish relevant and non-relevant recommendations. Pearson consistently outperforms Cosine in both precision and recall across all M values, showing that even though its MAE is higher, it is better at identifying relevant recommendations. This suggests that Pearson correlation captures more meaningful relationships between users and movies when evaluating the overall quality of recommendations, while Cosine similarity provides more accurate numerical rating predictions.

Another notable effect is on matrix density, which increases from 0.0858 (M=20) to 0.1737 (M=60). This confirms that filtering out movies with fewer ratings makes the dataset denser, meaning that a greater proportion of users have rated the remaining movies. This increased density improves recommendation reliability since more data is available for each remaining movie. Furthermore, as M increases, the number of movies in the dataset drops significantly, reducing from 1297 movies (M=20) to 335 movies (M=60), while the number of users remains mostly stable, slightly decreasing from 610 users (M=20) to 603 users (M=60). This suggests that filtering by M primarily removes less-rated movies without significantly affecting the user base.

Overall, higher M values lead to a denser dataset, slightly improved MAE, and better precision and recall, but at the cost of reducing the number of movies available for recommendation. The impact is more noticeable in Pearson similarity, which benefits from a denser dataset, whereas Cosine similarity consistently outperforms Pearson in terms of MAE across all M values, making it the better choice when filtering movies by rating count. However, Pearson achieves better recall and precision, meaning it generates more relevant recommendations.

Experiment Key	MAE	Macro Precision	Macro Recall	Matrix Density	Users	Movies	Ratings
Exp3 - No Movie Filter, M'=20 - N=90 - Cosine - predict_rating	0,6876	0,5835	0,5489	0,017	610	9724	100,836
Exp3 - No Movie Filter, M'=20 - N=90 - Cosine - predict_rating_popularity	0,6921	0,5749	0,5359	0,017	610	9724	100,836
Exp3 - No Movie Filter, M'=20 - N=90 - Cosine - predict_rating_low_popularity	0,6925	0,5745	0,5593	0,017	610	9724	100,836

Experiment Key	MAE	Macro Precision	Macro Recall	Matrix Density	Users	Movies	Ratings
Exp3 - No Movie Filter, M'=20 - N=90 - Pearson - predict_rating	0,7681	0,5955	0,6016	0,017	610	9724	100,836
Exp3 - No Movie Filter, M'=20 - N=90 - Pearson - predict_rating_popularity	0,7655	0,5922	0,5959	0,017	610	9724	100,836
Exp3 - No Movie Filter, M'=20 - N=90 - Pearson - predict_rating_low_popularity	0,7848	0,5931	0,5986	0,017	610	9724	100,836
Exp3 - No Movie Filter, M'=40 - N=90 - Cosine - predict_rating	0,6879	0,567	0,5364	0,0231	428	9660	95,711
Exp3 - No Movie Filter, M'=40 - N=90 - Cosine - predict_rating_popularity	0,689	0,5592	0,5257	0,0231	428	9660	95,711
Exp3 - No Movie Filter, M'=40 - N=90 - Cosine - predict_rating_low_popularity	0,6943	0,558	0,5455	0,0231	428	9660	95,711
Exp3 - No Movie Filter, M'=40 - N=90 - Pearson - predict_rating	0,7754	0,5846	0,5903	0,0231	428	9660	95,711
Exp3 - No Movie Filter, M'=40 - N=90 - Pearson - predict_rating_popularity	0,7688	0,5856	0,5894	0,0231	428	9660	95,711
Exp3 - No Movie Filter, M'=40 - N=90 - Pearson - predict_rating_low_popularity	0,794	0,5813	0,5861	0,0231	428	9660	95,711
Exp3 - No Movie Filter, M'=60 - N=90 - Cosine - predict_rating	0,6871	0,578	0,5477	0,0282	336	9618	91,14
Exp3 - No Movie Filter, M'=60 - N=90 - Cosine - predict_rating_popularity	0,6886	0,5652	0,5315	0,0282	336	9618	91,14
Exp3 - No Movie Filter, M'=60 - N=90 - Cosine - predict_rating_low_popularity	0,6929	0,5696	0,5582	0,0282	336	9618	91,14
Exp3 - No Movie Filter, M'=60 - N=90 - Pearson - predict_rating	0,7845	0,5921	0,5985	0,0282	336	9618	91,14
Exp3 - No Movie Filter, M'=60 - N=90 - Pearson - predict_rating_popularity	0,7749	0,5909	0,5962	0,0282	336	9618	91,14
Exp3 - No Movie Filter, M'=60 - N=90 - Pearson - predict_rating_low_popularity	0,8058	0,588	0,5923	0,0282	336	9618	91,14

Table 5: Impact of Filtering Users on Recommendation Performance

In Table 5, we observe the same trends in MAE, Macro Recall, and Macro Precision as we did in Table 4, where we filtered based on M (minimum ratings per movie). The Mean Absolute Error (MAE) remains relatively stable across different M' values, showing only small variations. Similarly, Precision and Recall patterns remain consistent, with Pearson similarity generally achieving better Precision and Recall than Cosine similarity, while Cosine similarity achieves slightly better MAE values. However, a key difference in this table is the significantly lower matrix density, which indicates that filtering users based on M' results in a much sparser dataset. As we increased M', we observed a

noticeable drop in the number of users, leading to a lower number of total ratings. For example, filtering with M'=60 reduced the dataset from 100,836 ratings to 91,140 ratings, and the density dropped from 0.0170 to 0.0282.

6 Results and Analysis

In this section, we present a comprehensive evaluation of the model performance across all the experiments conducted. We analyze the impact of different filtering strategies (M and M'), similarity measures (Cosine and Pearson), and rating prediction methods on the accuracy and effectiveness of recommendations. We summarize the key findings from our experiments, highlighting the best-performing configurations and discussing the trade-offs between different approaches.In addition we provide graphical representations of key metrics such as MAE, Macro Precision, and Macro Recall, visualizing trends and comparisons that were not explicitly showcased in previous sections. These visualizations will allow for a deeper understanding of how each filtering strategy and similarity measure influenced the recommendation system's accuracy and reliability.

6.1 Summary of Model Performance

Overall, across all the experiments conducted, we observed that the Mean Absolute Error (MAE) remained relatively stable, averaging around 0.7, while Macro Precision and Macro Recall stayed close to 0.6. Despite testing different filtering strategies (M and M'), similarity measures (Cosine and Pearson), and rating prediction functions, the variations in performance were relatively minor, with a maximum difference of approximately 0.1 for each metric.

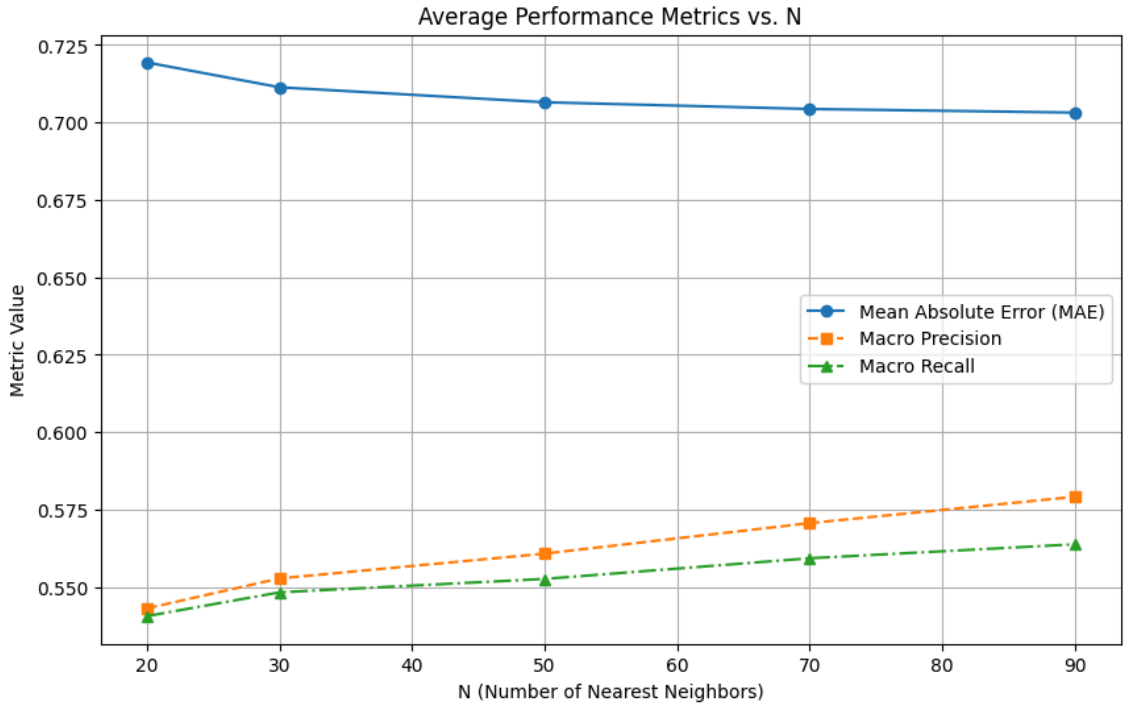
This suggests that while filtering strategies and similarity measures have some impact, the fundamental approach to recommendation (collaborative filtering with a user-based threshold) limits the extent of improvements. One important factor affecting the results is the way the model determines whether a movie is relevant to a user—by considering a movie as relevant if its rating is higher than the user's average rating. This threshold might not always reflect true user preferences, as some users might give generally high or low ratings. A possible improvement would be to adjust this threshold, for example, setting the relevance criterion as "average rating + 0.7" instead of just the average. This would allow the system to better differentiate truly liked movies from neutral ones.

Future enhancements could also include hybrid models, matrix factorization techniques (SVD, NMF), or deep learning-based approaches to better capture user preferences and improve recommendation accuracy.

6.2 Graphical Representation of Metrics

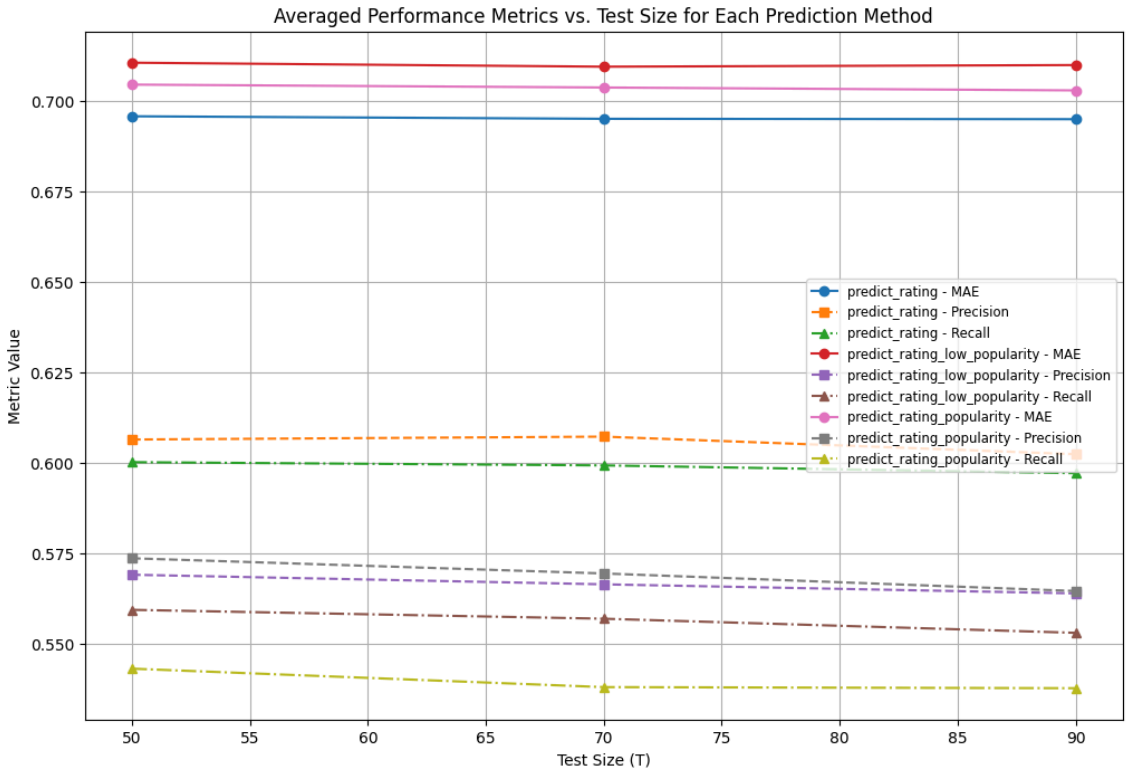
For each experiment conducted in this study, we will present visualizations to illustrate the performance metrics effectively. These graphical representations will include various plots that showcase key evaluation measures such as Mean Absolute Error (MAE), macro average precision, and macro average recall. By visualizing the results, we aim to provide a clear and intuitive understanding of the impact of different experimental settings on the recommendation system's performance. These plots will help compare different approaches and highlight trends in system accuracy under varying conditions.

The graph shown represents the results from the first experiment, where we analyze the system's performance across different values of NNN (the number of nearest neighbors). As NNN increases, we observe distinct trends in the plotted metrics, highlighting how the system responds to varying neighborhood sizes. This visualization helps in understanding the overall behavior of the recommendation model under these experimental conditions.



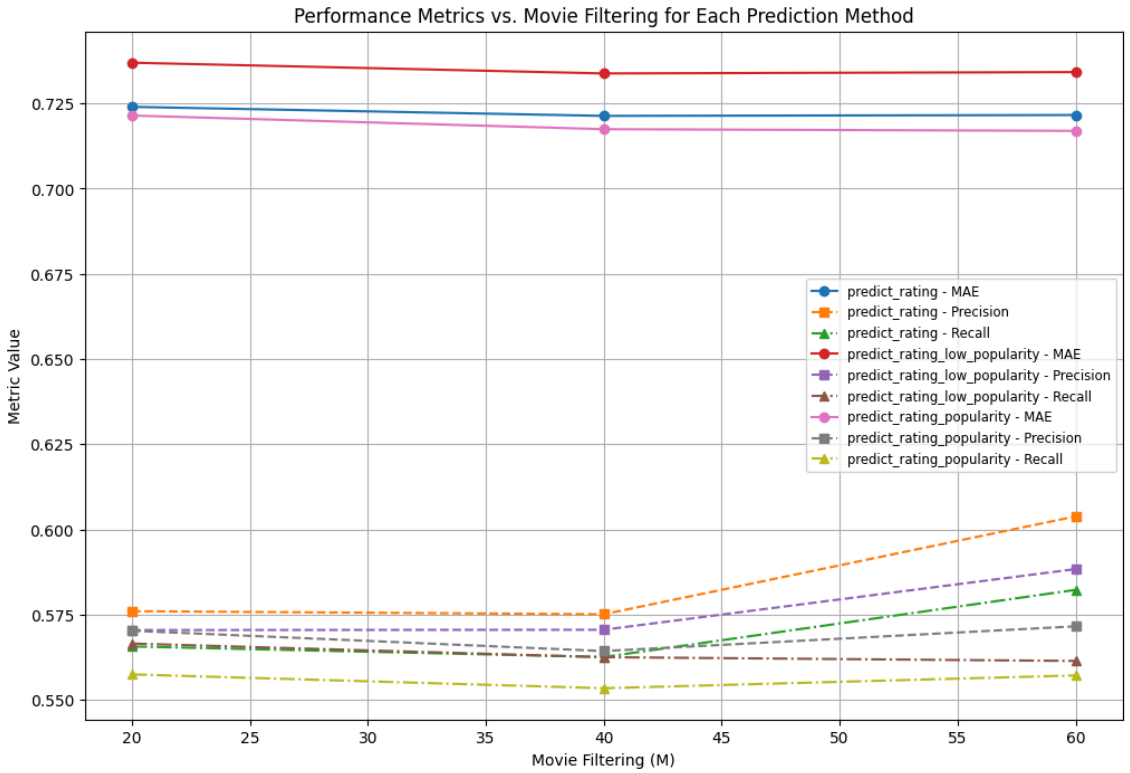
Plot 1: Average performance metrics vs N

The second graph shown represents the results from the second experiment, where we examine the system's performance across different test sizes (T). As T varies, we observe how different prediction methods behave under these conditions. The visualization provides insights into how the model adapts to different proportions of training and test data, helping to compare the impact of various prediction approaches.

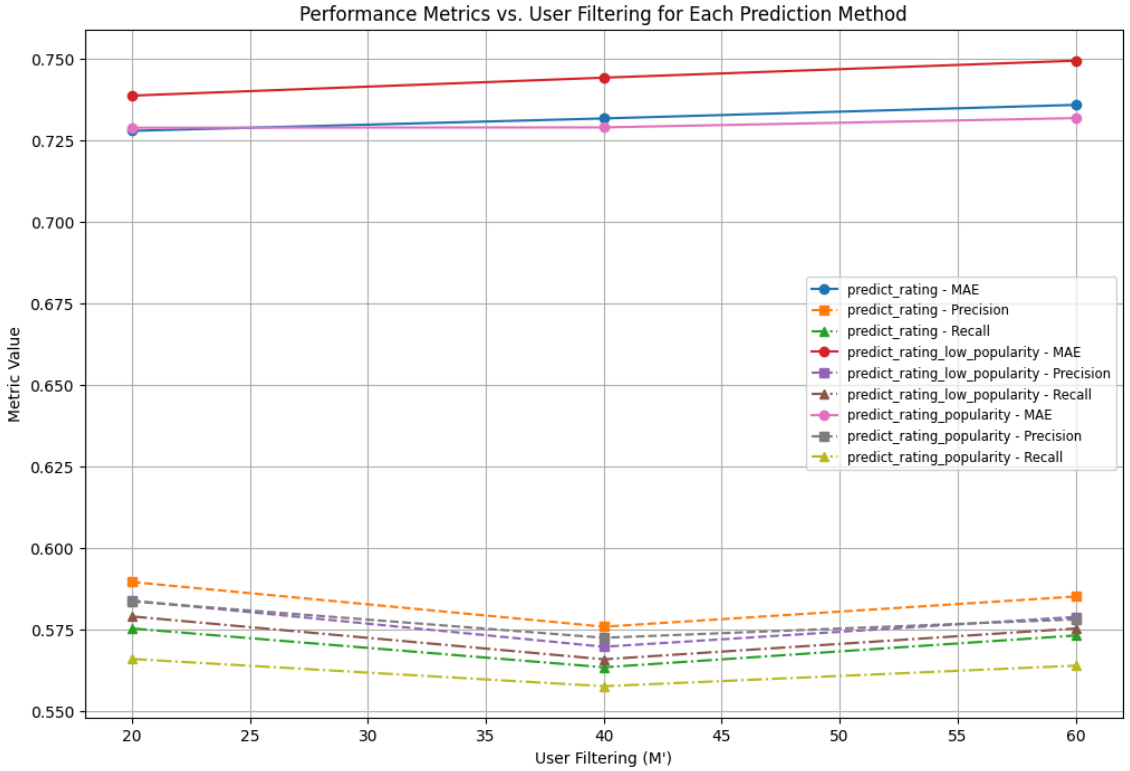


Plot 2: Average performance metrics vs Test size for each prediction method

The last two graphs correspond to the last experiment but depict different aspects of the filtering process. The first plot illustrates the effect of filtering movies based on the minimum number of ratings required (M), while the second plot examines the impact of filtering users based on their minimum number of ratings (M′). Both visualizations provide insight into how varying these thresholds influences the performance of different prediction methods, helping to assess the model's behavior under different data density conditions.



Plot 3: Average performance metrics vs Movie filtering for each prediction method



Plot 4: Average performance metrics vs User filtering for each prediction method