COMPARISON OF RECOMMENDATION ALGORITHMS ON THE MOVIELENS DATASET

QIN JIAYUE
A0294788B
Data Science and Mchine learning
National University of Singapore
e1348926@u.nus.edu

November 12, 2024

ABSTRACT

In response to the growing demand for personalized recommendations, this paper investigates and compares several recommendation algorithms on the MovieLens dataset. We evaluate content-based filtering, item-based collaborative filtering (IBCF), SVD-based, and Funk-SVD techniques, assessing each method's hit rate and runtime efficiency. Results indicate that item-based collaborative filtering achieves the highest hit rate, effectively capturing user preferences, while SVD-based methods demonstrate the best runtime efficiency. These findings highlight a trade-off between accuracy and efficiency, offering valuable insights for practical applications in recommendation systems.

***K*eywords** Collaborative filtering · Content-based filtering · Recommendation system

1 Introduction

The vast amount of information on the Internet has made information retrieval systems essential tools for guiding users to the content they seek. Users increasingly demand personalized search results tailored to their preferences, a role fulfilled by recommendation systems. By analyzing user profiles, these systems predict the relevance of specific items, proving valuable in fields such as e-commerce, with promising applications in other domains as well. Collaborative filtering, which suggests items based on the preferences of other users, is particularly popular, and various algorithms have been developed to improve recommendation accuracy. This paper aims to compare the performance of recommendation algorithms discussed in the course with other existing methods, examining their strengths and weaknesses.

The structure of this article is as follows. Section 2 introduces the datasets used, followed by a discussion of the recommendation algorithms and their advantages and disadvantages in Section 3. Section 4 presents the application of these algorithms to the dataset. Finally, we conclude the study and discuss potential future directions.

2 Dataset Description

2.1 Dataset Overview

This project utilizes the MovieLens 20M dataset [1] for evaluation. The MovieLens dataset is one of the most widely used resources among researchers and developers in the field of collaborative filtering. Its accessibility and extensive use in previous studies make it particularly relevant for our purposes.

The dataset contains a total of 20,000,263 ratings and 465,564 tag applications across 27,278 movies. These data were generated by 138,493 users between January 9, 1995, and March 31, 2015, with all included users having rated at least 20 movies. Due to project constraints limiting the number of datasets to two, as well as limitations on time and computational resources, we selected 'rating.csv', which contains user ratings for movies, and 'movie.csv', which provides basic movie information. Details for each dataset are presented in Tables 1 and 2.

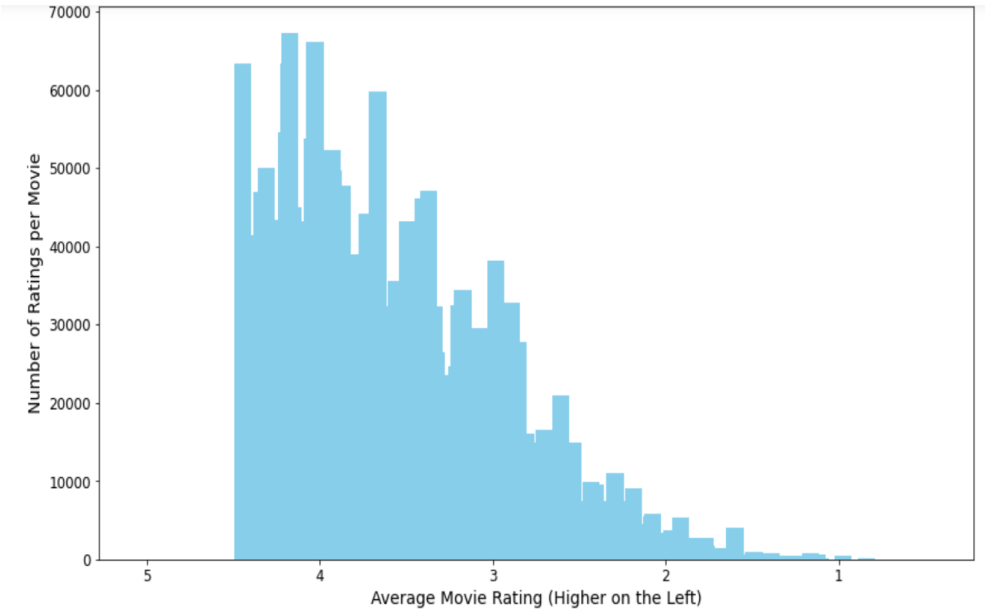


Figure 1: Long-tail effect in the MovieLens dataset

Table 1: Columns in ratings.csv

Column Name	Data Type	Description
userId	int64	Unique identifier for each user who provided a rating.
movieId	int64	Unique identifier for each movie rated by users, matching IDs in the movies dataset.
rating	float64	Rating score given by a user to a movie, stored as a floating-point number.
timestamp	object	Time when the rating was given, typically a Unix timestamp.

Table 2: Columns in movies.csv

Column Name	Data Type	Description
movieId	int64	Unique identifier for each movie, matching IDs in the ratings dataset.
title	object	Title of the movie, including its name and release year.
genres	object	Genre(s) associated with each movie.

Neither of datasets contain no missing values. In the ratings dataset, there are 138,493 unique users who provided ratings for 26,744 unique movies, while the movies dataset contains information on 27,278 movies. The top-rated movies include "Only Daughter" and "Rocaterrania."

As illustrated in Figure 1, movies with high average ratings generally receive more ratings, while movies with lower ratings tend to have fewer interactions. This phenomenon is typical in movie datasets, where popular movies attract more user attention and consequently receive more ratings.

Figure 2 provides a word cloud visualization of the genres present in the dataset. The font size of each genre label in the word cloud reflects its frequency, with larger fonts indicating genres that are more common. This visualization offers an intuitive overview of the genre distribution, highlighting popular categories such as Drama, Comedy, and Thriller.

As shown in Figure 3, the distribution of ratings per user indicates that most users rate only a small number of movies. This distribution reflects a common challenge in recommendation systems, where sparse user interactions can limit the effectiveness of collaborative filtering and create "cold start" issues for new or infrequent users.

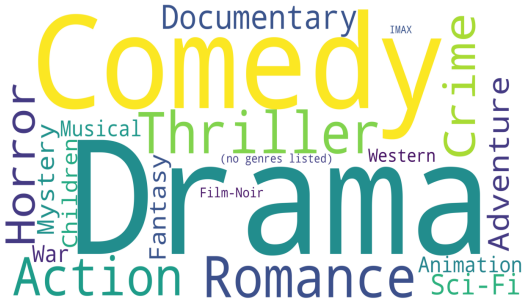


Figure 2: Genre frequency in the MovieLens dataset

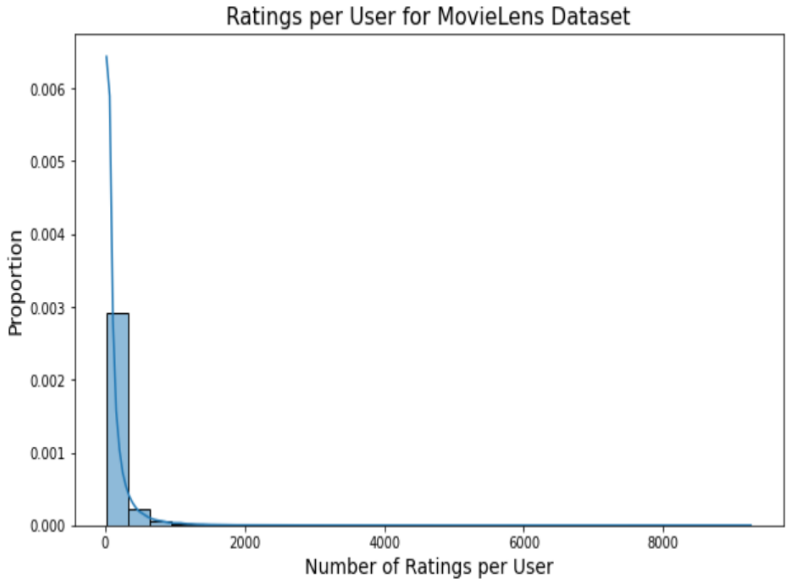


Figure 3: Distribution of Ratings per User in the MovieLens dataset

2.2 Data Preprocessing

Given the constraints on runtime and computational resources, we sampled a subset of 10,000 users based on the original distribution of ratings per user, as illustrated in the Figure 3. This sampling strategy ensures that the selected subset retains the characteristics of the overall dataset, preserving the variability in user rating behavior for analysis. To facilitate subsequent computations, we extracted the release year of each movie from the 'title' column, creating a new feature 'year'. This feature provides additional temporal context that may be valuable in understanding user preferences over time. Finally, we merged the ratings data of the sampled users with the movie information from the 'movies' dataset. This resulted in a preprocessed dataset, which will be used in the following analyses. In the preprocessed dataset, the maximum number of movies rated by a single user is 9,254, while the minimum is 20. Additionally, the most-rated movie received 7,167 ratings, whereas some movies were rated by only one user.

2.3 Visualizations

The preprocessed dataset contains 4,365,121 rows and 7 columns: `UserId`, `movieId`, `rating`, `timestamp`, `title`, `genres`, and `year`. To better understand the distribution of ratings, we generated a histogram, as shown in Figure 4. This distribution reveals that users tend to rate movies they enjoyed, as evidenced by the higher frequency of ratings 3, 4, and 5 compared to ratings 1 and 2. The observed pattern indicates a positive bias in user ratings, where higher ratings are more common. This skew toward favorable ratings suggests that, when calculating similarity based on ratings, it is beneficial to normalize the ratings by subtracting the user's average rating. This mean normalization helps

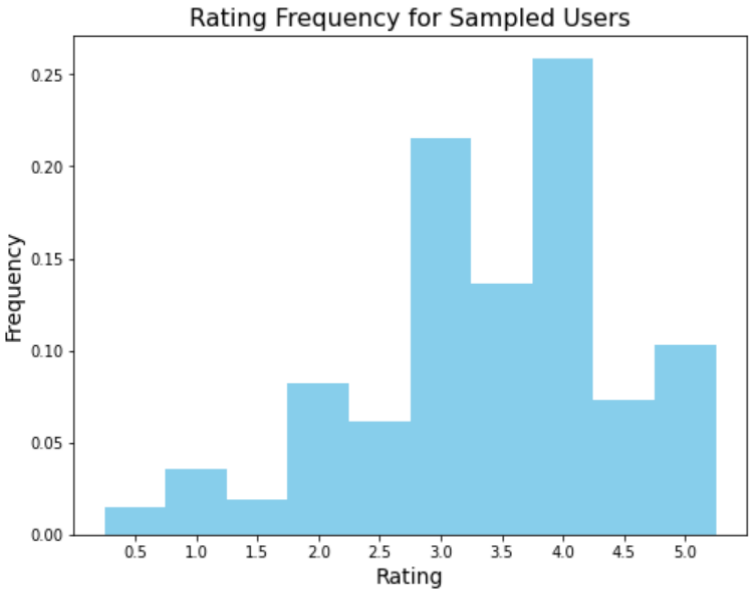


Figure 4: Rating frequency for sampled users

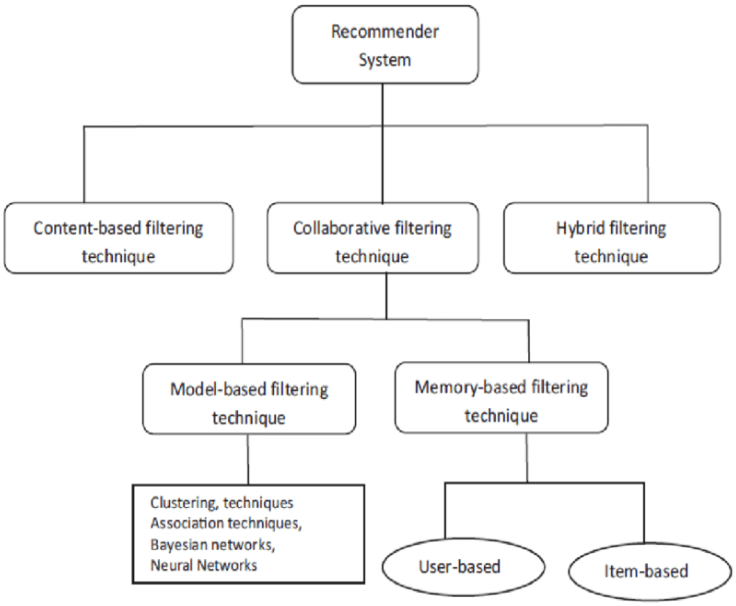


Figure 5: Types of recommendation system

reduce the bias introduced by individual rating tendencies, providing a more accurate basis for similarity calculations in recommendation models.

3 Methodology

Since the inception of recommender systems, extensive research has advanced the field. Much of the existing literature also highlights areas for future exploration. Figure 5 presents an overview of different types of recommender systems. In this section, various approaches to recommendation are outlined as part of related work.

3.1 Content-based Filtering Technique

Content-based filtering recommends items by analyzing features of those a user has previously engaged with, such as ratings or shares, to build a personalized profile. Recommendations are made by comparing this user profile to other items, suggesting those with the highest similarity. This approach is independent of other users, avoiding the cold-start problem and enabling recommendations for unique tastes and new items. It is also explainable, as recommendations are based on the user profile's content features. However, there are limitations. Identifying relevant features can be challenging, especially for complex items like movies. It also struggles with recommending to new users, as profiles rely on past interactions. Additionally, content-based filtering may lead to overspecialization, limiting diversity, and cannot incorporate peer quality judgments.

The steps of this method are as follows:

Step 1: Constructing the Item Profile

For movie recommendations, each item's profile is defined by its genre keywords. Since it is unnecessary to penalize frequently occurring words within genres, we use simple word counts rather than term frequency-inverse document frequency (TF-IDF). Using the `CountVectorizer` class, we transform the genre text into a frequency matrix, where each element $w_{ij} = a[i][j]$ represents the frequency of the j^{th} word in the i^{th} genre text.

Step 2: Building the User Profile

User profiles are generated by weighting item profiles based on the user's average rating. For a user x who has rated four movies with scores r_1, r_2, r_3, and r_4, we compute the mean rating $\bar{r}$. The user profile $v(x)$ is then derived as follows:

$$v(x) = \frac{1}{4}\left[(r_1 - \bar{r}) \cdot g_1 + (r_2 - \bar{r}) \cdot g_2 + (r_3 - \bar{r}) \cdot g_3 + (r_4 - \bar{r}) \cdot g_4\right] \tag{1}$$

where g_1, g_2, g_3, and g_4 are the profiles of the rated movies, and each term reflects the deviation from the user's mean rating, enhancing the personalization of the user profile.

Step 3: Calculating Similarity between User and Item Profiles

Finally, to generate recommendations, we calculate the cosine similarity between the user profile and each item profile. Cosine similarity, defined as follows, is widely used in content-based filtering due to its effectiveness in capturing the angular similarity between two vector representations:

$$\text{similarity}(v(x), g) = \frac{v(x) \cdot g}{\|v(x)\|\|g\|} = cos\theta \tag{2}$$

This approach yields a list of items ranked by their similarity scores, from which the most similar items are recommended to the user.

3.2 Collaborative Filtering Technique

Collaborative Filtering is a recommendation technique based on users' past behavior rather than contextual information. It relies on identifying users with similar preferences, tastes, and choices to generate personalized recommendations. By analyzing patterns in users' behavior, such as ratings and activity data, collaborative filtering predicts items a user might like based on their similarity to other users. Collaborative filtering utilizes rating data from multiple users across various items to estimate missing ratings and generate a top-N recommendation list of the most popular items for a specific user, known as the active user. This study primarily focuses on methods introduced in the course. The user-based filtering technique was excluded from consideration due to its high computational cost, which is mainly attributed to the need for calculating similarity scores between each pair of users. Additionally, in real-world applications, user characteristics tend to be complex and frequently change over time, making it challenging to maintain accurate user profiles for reliable recommendations. Consequently, we opted not to use the user-based filtering approach in this study.

3.2.1 Item-based Filtering Technique

Item-based algorithms recommend items that are similar to those a user has previously liked or interacted with. This approach focuses on finding similarities between items rather than users, making it particularly useful in scenarios where user preferences may be complex or frequently changing. Item-based filtering tends to be more stable than user-based filtering, as item properties usually remain consistent over time. However, it has limitations, including

the cold start problem, which needs substantial user or item data to generate accurate recommendations, limiting its effectiveness for new or niche items with few ratings. The sparsity of the utility matrix can also reduce performance, as it's difficult to find users with overlapping ratings. Additionally, collaborative filtering often exhibits popularity bias, favoring popular items over diverse recommendations, which may not satisfy users with unique tastes.

The specific steps of the item-based filtering process are as follows:

Step 1: Constructing the User-Item Rating Matrix

First, a user-item rating matrix R is created based on user ratings of items, where r_{xj} represents the rating given by user x to item j.

Step 2: Calculating Similarity between Items

The similarity between items i and j is calculated using the Pearson correlation coefficient between their respective rating vectors. The $\text{sim}(i, j)$ is defined as:

$$\text{sim}(i,j) = \frac{\sum_{x \in U}(r_{xi} - \bar{r}_i) \cdot (r_{xj} - \bar{r}_j)}{\sqrt{\sum_{x \in U}(r_{xi} - \bar{r}_i)^2} \cdot \sqrt{\sum_{x \in U}(r_{xj} - \bar{r}_j)^2}} \tag{3}$$

where U is the set of users who have rated both items i and j, $\bar{r}_i$ is the average rating of item i across all users in U, and $\bar{r}_j$ is the average rating of item j across all users in U. This similarity score quantifies how closely related two items are based on user ratings.

Step 3: Predicting the Rating of an Item

To predict the rating of an item i for a user x, we aggregate the ratings of items similar to i that user x has already rated. This predicted rating r_{xi} is calculated as follows:

$$r_{xi} = \frac{\sum_{j \in N(i;x)} \text{sim}(i,j) \cdot r_{xj}}{\sum_{j \in N(i;x)} |\text{sim}(i,j)|} \tag{4}$$

where $N(i; x)$ represents the set of items rated by user x that are similar to item i. This weighted average takes into account the similarity between items, emphasizing ratings of items that are more similar to the target item.

3.2.2 Support Vector Decomposition (SVD)

Support Vector Decomposition (SVD) is a model-based collaborative filtering technique based on matrix factorization (MF), commonly used for latent variable decomposition and dimensionality reduction in recommendation systems. SVD was first applied to collaborative filtering by Billsus and Pazzani in 1998, who identified it as an effective method for recommendation tasks [2]. The formula for matrix factorization with SVD is:

$$R = U\Sigma V^T$$

where R is an $m \times n$ user-item rating matrix, with m as the number of users and n as the number of items (e.g., movies). This decomposition yields three matrices: U, Σ, and V^T. Here, U is an $m \times r$ matrix representing user embeddings, with r latent factors. Each row of U reflects a user's preferences, while Σ, a $r \times r$ diagonal matrix, contains the singular values of R, indicating each latent factor's strength. V^T is an $r \times n$ matrix representing item embeddings, with rows corresponding to item characteristics.

The goal of SVD is to find optimal values for U, Σ, and V^T such that $U\Sigma V^T$ closely approximates R. This is achieved by minimizing reconstruction error, often using optimization techniques like least squares or gradient descent. By retaining only the most K significant singular values, SVD captures essential latent factors that drive user preferences and item characteristics.

After matrix factorization, missing ratings in R can be estimated by taking the dot product of corresponding rows of U, Σ, and columns of V^T. These estimated ratings are used to generate personalized recommendations, suggesting items with the highest predicted ratings.

3.2.3 Funk-SVD Algorithm

Traditional SVD requires significant storage for the complete matrix and has high computational complexity. To address these issues, Simon Funk introduced Funk-SVD in 2006 during the Netflix Prize competition [3]. Unlike traditional

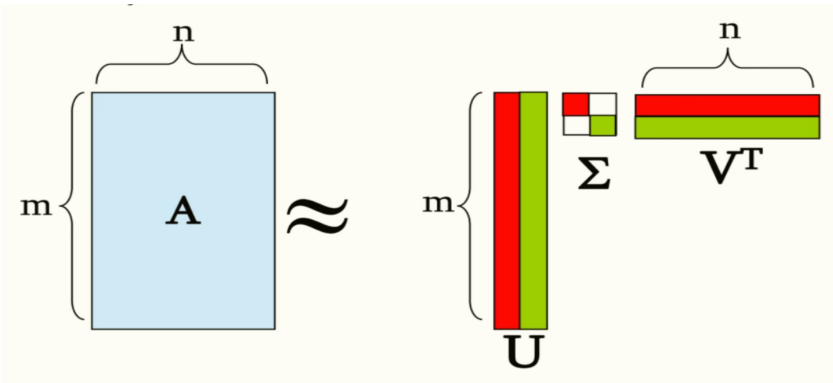


Figure 6: Matrix decomposition using SVD

SVD, Funk-SVD does not rely on heuristic similarity but instead employs an explicit learning process. Funk-SVD is also referred to as a latent factor model (LFM).

In Funk-SVD, we make no assumptions about matrices P and Q; they are simply two factor matrices whose product approximates the original rating matrix. Additionally, the value of f (number of latent factors) is not subjectively chosen. Instead, we decompose the rating matrix R into the product of two lower-dimensional matrices, $P \in \mathbb{R}^{f \times m}$ and $Q \in \mathbb{R}^{f \times n}$. The objective is to minimize the error on training data to learn the optimal values for P and Q. The loss function $C(p, q)$ includes two components: squared loss and regularization.

The loss function can be defined as follows:

$$C(p,q) = \sum_{(x,i) \in \text{Train}} \left(r_{xi} - p_x^\top q_i\right)^2 + \lambda \left(\|p_x\|^2 + \|q_i\|^2\right) \tag{5}$$

where r_{xi} is the observed rating of user x for item i, p_x and q_i are the latent vectors for the user and item respectively, and λ is the regularization parameter to prevent overfitting.

Algorithm 1 Funk-SVD Algorithm

Initialize P and Q:
$p_{xk}, q_{ik} \leftarrow \text{random}()/\sqrt{f}$
for step = 1 to n **do**
  for (x, i, r_{xi}) in train.items() **do**
    $\text{err}_{xi} \leftarrow r_{xi} - \langle p[x, : f+1], q[i, : f+1] \rangle$
    for k = 1 to f **do**
      $p_{xk} \leftarrow p_{xk} + \alpha \cdot (q_{ik} \cdot \text{err}_{xi} - \lambda \cdot p_{xk})$
      $q_{ik} \leftarrow q_{ik} + \alpha \cdot (p_{xk} \cdot \text{err}_{xi} - \lambda \cdot q_{ik})$
    end for
  end for
end for
return P, Q

3.3 Evaluation Metrics

The metrics used to assess the accuracy of a recommendation algorithm can be broadly categorized into two types: statistical accuracy metrics and decision-support metrics [4]. In this paper, hit rate is selected to evaluate the decision-support accuracy of the recommendation algorithm.

The hit rate is calculated as follows:

$$\text{Hit Rate} = \frac{\text{Number of Hits}}{\text{Total Number of Recommendations}} \tag{6}$$

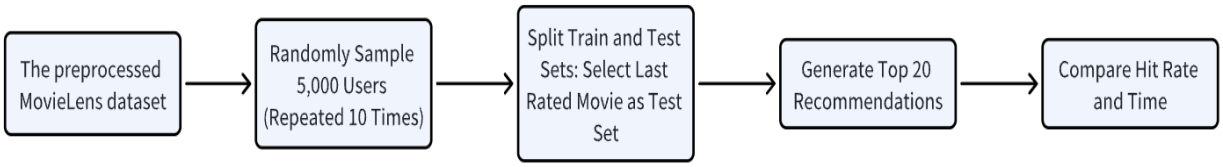


Figure 7: Workflow

where a "hit" indicates a recommended item that matches an item actually chosen or interacted with by the user. This metric reflects the algorithm's ability to present relevant items within the recommendations provided to the user.

4 Experiments and Results

The experiments were conducted using the preprocessed MovieLens dataset, which contains rating data from 10,000 users. In order to compare the performance of different recommendation algorithms, we randomly sampled 5,000 users and repeated this process 10 times. For each user, the last rated movie was selected as the test set, while the remaining ratings were used as the training set. For each user, we recommended the top 20 movies, recording the hit rate and the time taken to generate recommendations in each experiment. Figure 7 provides an overview of the experimental workflow.

In the implementation of the content-based recommendation algorithm, we utilized `CountVectorizer` to precompute the term frequencies for movies rated by 10,000 users, generating item profiles and avoiding redundant computations in subsequent sampling. Based on the similarity calculation described in Section 3.1, we then computed the similarity between each user and unrated movies, selecting the top 20 movies to recommend to each user. For the item-based collaborative filtering (IBCF), SVD-based, and Funk-SVD techniques, we constructed a user-item rating matrix for each experiment. Using the methodology outlined in Section 3.2, we estimated each user's rating for unrated movies to generate personalized recommendations.

In Figure 8(a), the performance comparison of the chosen techniques is shown. Among these, the item-based collaborative filtering (IBCF) method outperforms the others in terms of hit rate, indicating a higher accuracy in capturing user preferences. The content-based method, however, performs the worst, primarily due to its reliance on item features rather than collaborative user information, which limits its ability to generalize beyond user-specific interests.

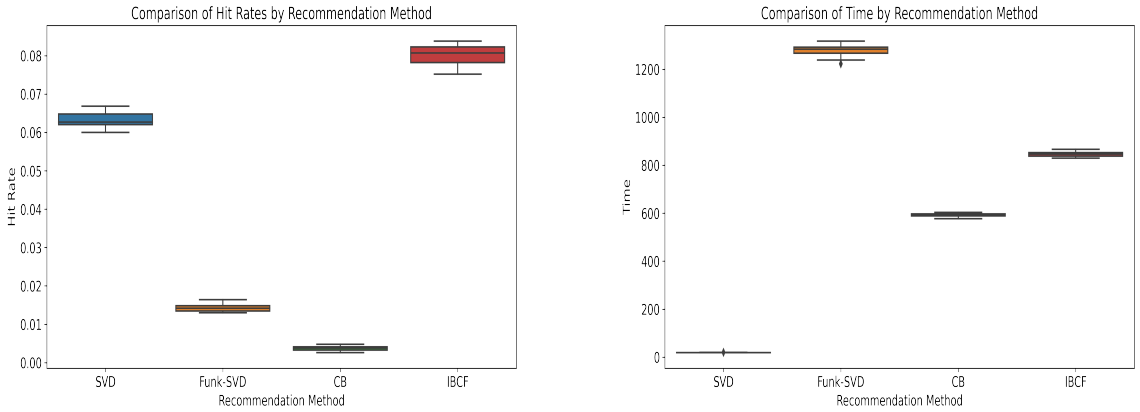
Figure 8(b) compares the runtime of different recommendation algorithms for predicting user preferences for movies. As shown, the SVD-based method achieves the highest efficiency, with significantly lower computation times compared to the other techniques. Funk-SVD, due to its iterative optimization process, exhibits the longest runtime. The content-based and IBCF methods fall between these two, with moderate time requirements.

The experiments were conducted in a Python 3.6 environment. Python code link is : `https://github.com/qjymary/COMPARISON-OF-RECOMMENDATION-ALGORITHMS-ON-THE-MOVIELENS-DATASET/blob/main/DSA5101project-python-20241112.ipynb`

5 Conclusion and Discussion

This paper utilized several recommendation algorithms introduced in the course, including content-based, item-based collaborative filtering (IBCF), SVD-based, and Funk-SVD techniques, to assess their effectiveness on the MovieLens dataset. Through our experiments, we observed distinct performance characteristics for each method, reflecting their theoretical foundations and practical applications in recommendation systems.

Our findings indicate that IBCF outperformed the other techniques in terms of hit rate, suggesting it more accurately captures user preferences by leveraging collaborative information among users. This result aligns with our expectation that collaborative filtering can better generalize user preferences by considering other users with similar interests. In contrast, the content-based method exhibited the lowest hit rate. This result likely stems from its reliance solely on item features, which limits its capacity to understand complex user interactions and adapt to user preferences that extend beyond explicitly rated items. In terms of runtime, the SVD-based method proved to be the most efficient, making it a viable choice for large-scale, real-time recommendation systems. This efficiency comes from the reduced



(a) Hit Rate Comparison

(b) Runtime Comparison

Figure 8: Comparison of hit rates and runtime by recommendation method

dimensionality of the user-item matrix, allowing faster computations. On the other hand, Funk-SVD exhibited the longest runtime, likely due to the iterative nature of its optimization process, which demands significant computational resources. The IBCF and content-based methods demonstrated moderate time requirements, indicating a trade-off between accuracy and efficiency in practical applications.

In addition, we attempted to use the content-based filtering technique to fill in the user-movie rating matrix to address matrix sparsity and the cold start problem. We then applied item-based filtering on the filled matrix. However, the results did not show any improvement over using item-based filtering alone, so these findings are not presented in this paper. The lack of improvement in hit rate warrants further investigation.

Future work could focus on enhancing the content-based model by incorporating collaborative signals or leveraging deep learning techniques to capture more complex relationships between users and items. These potential improvements highlight the continuous evolution of recommendation systems and underscore the importance of adapting techniques to meet the specific demands of diverse applications."

References

[1] Alemi, F., Jackson, J., Kessler, R., & Takeuchi, D. (2016). *MovieLens 20M Dataset* [Data set]. Kaggle. https://www.kaggle.com/datasets/grouplens/movielens-20m-dataset/data

[2] Billsus, D., & Pazzani, M. J. (1998). Learning collaborative information filters. In *Proceedings of the 15th International Conference on Machine Learning* (pp. 46–54). Morgan Kaufmann.

[3] Funk, S. (2006). Netflix update: Try this at home [Blog post]. *Sifter*. http://sifter.org/s̃imon/journal/20061211.html

[4] Herlocker, J. L., Konstan, J. A., Borchers, A., & Riedl, J. (1999). An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 230–237). ACM. https://doi.org/10.1145/312624.312682