Proceedings of the 5th International Conference on Computing and Data Science
DOI: 10.54254/2755-2721/14/20230791

Research and improvement of movie recommendation based on a collaborative filtering algorithm

Junxiang Zhao

Southwest University, No. 2, Tiansheng Road, Beibei District, Chongqing

547616493@qq.com

Abstract. Among various recommendation algorithms, collaborative filtering stands out as a popular and effective technique that analyzes user behavior to suggest personalized items. The goal of studying recommendation algorithms is to enhance the exactness of recommendation outcomes and provide users with tailored suggestions. In this paper, the algorithm of collaborative filtering is divided into User-CF and Item-CF, and their algorithm principles and implementation steps are introduced. The similarity calculation of both methods is improved, and a comparison of their advantages and disadvantages is conducted. Experiments are conducted on the MovieLens-latest-small movie dataset to compare the performance of User-CF and Item-CF before and after their similarity calculation is improved. The results show that the similarity improvement of User-CF can substantially enhance the algorithm's inclusiveness, while the similarity improvement of Item-CF does not have a considerable impact on the algorithm's performance indicators. Finally, the comparison shows that Item-CF has better coverage and popularity, while User-CF has better precision and recall.

Keywords: collaborative filtering, recommendation algorithm, User-CF, Item-CF, similarity calculation, movie recommendation.

1. Introduction

The increasing significance of recommendation systems is attributed to the swift expansion of online businesses. Some applications heavily rely on recommendation systems to bring significant revenue improvements, such as food delivery platforms, short video platforms, music platforms, and shopping platforms. The results of a recommendation system will affect users' decisions in these applications, thus impacting their experience of these applications. Traditional recommendation algorithms primarily comprise content-based, collaborative filtering, and mixed recommendation methods. Among the various recommendation techniques, collaborative filtering has emerged as a popular and effective approach [1]. The primary aim of a recommendation system is to provide users with recommendations that cater to their wants and predilections. Therefore, the core of personalized recommendation is how to accurately obtain users' interests and predict their preferences for items [2].

In general, designing a recommendation system algorithm involves two core issues: the user and item embedding representation problem and the user-item interaction prediction modeling problem [3]. Collaborative filtering constructs a graph from user-item historical interaction data and then embeds users and items to generate feature vectors using algorithms such as matrix factorization. The user-item interaction can be modeled by utilizing the inner product [4]. The fundamental core of collaborative

DOI: 10.54254/2755-2721/14/20230791

filtering recommendation algorithms is "seeking similarity", which is, by modeling the historical interaction data between the current user and items, finding other users similar to the current user and recommending items that other users like to the current user (which is User-CF), and finding similar items to the items that the current user is provided recommendations based on their past interactions with the items (which is Item-CF). The success of this approach largely hinges on the accurate computation of similarity, which is a vital factor in determining the algorithm's effectiveness.

This study puts forward an approach to compute and enhance the similarity of User-CF and Item-CF, and conducts a comprehensive analysis of their respective pros and cons. This paper is divided into five parts. Part 2 will review the research status of optimization and the improvement of collaborative filtering. Part 3 will elaborate on the algorithm principles, implementation steps, similarity improvement, and comparison of User-CF and Item-CF in detail. Part 4 will conduct a comparative experiment based on the similarity before and after improvement and discuss the experimental results in depth. Finally, Part 5 will summarize the paper and discuss the findings, limitations, and future research directions of this study.

2. Literature review

To overcome the challenges of sparse data and cold-start situations in customary collaborative filtering algorithms, as well as to optimize their deficiencies in UI interaction, diversified intentions, preference information, temporal characteristics, semantic information, binary decision-making, and so on, scholars have proposed many improved algorithm models based on collaborative filtering.

Zhang et al. introduced a technique called neural collaborative filtering to enhance social recommendation algorithms, utilizing graph attention as a key component. They first pointed out that collaborative filtering still has data sparsity and cold-start problems. Some new methods that stack multiple GNN layers to obtain high-order social relationships and achieve good experimental results may be affected by the problem of excessive smoothing caused by stacking too many GNN layers, leading to a significant decline in performance. Then, based on two different graphs of user-item interaction and social network, Zhang et al. learned and modeled the latent factor representation of items and users, respectively. Afterwards, the latent factors of items and users were input into the neural collaborative filtering recommendation module, and by leveraging the graph's high-order connectivity, the embedding information was iteratively propagated on the graph neural network to enhance the node representation in the embedding space. Finally, the deep and complex interaction between items and users was captured to make predictions and recommendations [1].

Xia et al. conducted collaborative filtering with heterogeneous neighborhood aggregation (HNACF). They first pointed out that collaborative filtering algorithms use feature vectors generated by one-hot encoding, which have a small amount of information, and only dig out the connections between different behaviors of heterogeneous behavioral data, ignoring the connections between user behaviors. Then, based on the efficient heterogeneous collaborative filtering model EHCF, Xia et al. combined the characteristics of graph neural networks (GNN) to obtain HNACF [2].

Liu et al. introduced a convolutional memory graph collaborative filtering recommendation algorithm (CMGCF). Liu et al. first pointed out the problem of the decay of high-order collaborative information in the transmission between distant nodes. Then, Liu et al. reduced the loss of information transmission between nodes by introducing gated recurrent units in GNN, which can capture more complete high-order neighbor information in node embedding vectors. The embedding vectors of different layers were fused using a convolutional neural network to learn the interaction features between node vector dimensions in different time periods, thereby making the node vectors contain more preference information [3].

Liu et al. proposed an exercise recommendation algorithm that combines collaborative filtering and classification. Liu et al. first pointed out that collaborative filtering cannot make targeted recommendations for courses and exercises based on student interests and knowledge gaps. Then, Liu et al. integrated the exercise classification model with the collaborative filtering algorithm. Specifically, by recording various indicators of exercises completed by students, the exercises were classified by

DOI: 10.54254/2755-2721/14/20230791

difficulty, and a classification model was formed to determine the students' mastery of knowledge points. Together with the collaborative filtering algorithm, exercise recommendations were made [4].

Jiang et al. proposed a collaborative filtering algorithm for graph neural networks based on fusion meta-path. They first identified that the traditional collaborative filtering approach does not fully consider the interaction information between items and users and faces challenges such as data sparsity and cold-start problems. They then introduced meta-paths into the graph neural network collaborative filtering algorithm, starting with embedding user and item historical interactions into a bipartite graph and propagating multi-layer neural networks to obtain high-order features for users and items. They then used meta-path-based random walks to obtain latent semantic information from heterogeneous information networks. Finally, they fused the high-order features and latent features of items and users to make rating predictions [5].

A recommendation model called GCN-ONCF, which integrates graph convolution and cross product modeling, was introduced by Su et al. in their study on collaborative filtering. They first pointed out the limitations of traditional collaborative filtering methods, such as being affected by data cold-start and sparsity and having poor interpretability and generalization ability. They then used a convolutional neural network to model the UI interaction information instead of the dot product, explicitly capturing the pairwise relationships between embedding dimensions, resulting in the GCN-ONCF model [6].

Song et al. conducted a graph-neural collaborative filtering approach based on self-attention (UAGNCF). They first pointed out that the interaction coding of collaborative filtering signals in collaborative filtering algorithms necessitates a substantial quantity of data, making it difficult to effectively extract the necessary collaborative signals. Then, Song et al. applied the self-attention mechanism to the message construction between items and users, and then embedded the processed messages into the graph neural network layer with high-order connectivity, and finally aggregated the obtained messages before outputting [7].

Gao et al. introduced a bidirectional collaborative filtering recommendation algorithm using graph convolutional networks. They first pointed out that high-order nodes in graph neural network methods carry noise or negative information during information propagation. When the network depth reaches a certain level, the aggregation of node features for too many high-order nodes leads to the indistinguishability of different categories, which results in a decrease rather than an increase in the model's learning ability. Then, Gao et al. divided the original large-scale graph into multiple subgraphs for fusion learning and used the attention mechanism to optimize the combination of multiple node representations [8].

Yao et al. first conducted that the classic collaborative filtering algorithm in recommendation systems suffers from the problems of data sparsity and cold starts, which directly affect the accuracy of rating prediction. Then, they represented the original rating matrix as a user-item bipartite graph and used graph convolutional encoders to iteratively aggregate neighbor node information to obtain latent vector representations of users and items. The representations were then passed through a non-linear transformation layer to reconstruct the adjacency matrix, thus alleviating the impact of data sparsity on rating prediction [9].

Chen et al. introduced a graph convolutional collaborative filtering recommendation algorithm using the time series features. They first pointed out that the recommendation algorithm of graph convolutional neural networks did not focus on the temporal characteristics of user-item interaction embedding vectors. When and how many times users and items interacted were not used as information to participate in signal transmission. Then, Chen et al. rebuilt multiple datasets, retained original information such as temporal features in the datasets, summarized and extracted the historical temporal information of user-item interaction in the datasets, and parametrically processed it as an important feature input for the high-order collaborative signal transmission in the graph convolutional network model training [10].

Li et al. conducted a group recommendation method with a Nash equilibrium strategy and neural collaborative filtering. The core problem in group recommendation, as pointed out by Li et al., is the fusion of group members' preferences. Traditional fusion strategies mostly belong to single-type strategies, which can only satisfy the overall preference needs of the group to a certain extent. Some

DOI: 10.54254/2755-2721/14/20230791

users may always receive unsatisfactory (painful) recommendations, making it difficult to ensure the fairness of group recommendations. As the number of group members increases, the average satisfaction of group recommendations may decrease. Li et al. then jointly considered collaborative filtering methods and preference fusion strategies, proposing a group recommendation method that combines the Nash equilibrium strategy and neural collaborative filtering. On the one hand, inspired by non-cooperative game theory, a method is introduced that combines user preferences using a Nash equilibrium-based approach, where the aim is to solve the problem of maximizing group satisfaction as a Nash equilibrium problem, effectively alleviating the problem of preference conflicts among group members. On the other hand, under the action of deep neural networks, a collaborative filtering recommendation model integrated LFM and MLP is implemented based on neural networks, achieving linear and nonlinear interactions between latent feature vectors of users and items [11].

Chen et al. introduced a deep neural network matrix factorization recommendation algorithm using similarity calculation (SeqDMF). Chen et al. first proposed that most of the previous similarity algorithms focus on the rating levels of users for items and compare distances in vector form, ignoring the meaning of user behavior as a regular sequence. Moreover, the degree of the user's tendency towards items will change according to time. When the user's demand for the recommended results is small, it is necessary to place the items that the user is most interested in as close to the front as possible. Then, Chen et al. treated user information as a class sequence and used the longest common subsequence algorithm to find similar users, making the recommended results that are more in line with the user's recent interests appear earlier, making the hidden feature vectors between similar users closer, and maintaining this relationship when using deep neural networks to learn the hidden features between users and items, thereby improving the predictive accuracy of the model [12].

Zhang et al. conducted a neural graph for collaborative filtering using interaction intent. The authors first pointed out the shortcomings of traditional collaborative filtering and graph neural network recommendation algorithms, specifically the coarse granularity of user-item relationships and the neglect of diverse user intents when selecting items. Zhang et al. then proposed the interactive intent-integrated graph neural network collaborative filtering algorithm (INTNGCF), which uses a CF-PoolFormer attention mechanism in the latent intent modeling layer to decompose edges into multiple latent spaces and identify potential intents. The fusion layer then uses CF-PoolFormer attention to determine the importance of these intents and combines them to obtain a unified UI embedding. Finally, the model uses the CF-PoolFormer rating prediction method to predict scores based on the concatenated embeddings [13].

Cheng et al. introduced a personalized recommendation incorporated sequential three-way decisions with a single feed-forward neural network. (STWD-SFNN-PR). The authors first pointed out that most collaborative filtering and its improvement models do not consider features such as user and item characteristics, making it difficult to explore the nonlinear relationships between samples. Additionally, the topology structure of deep learning networks is based on binary decision trees, which neglects the difficulty level of recommendation samples. Cheng et al. then used embeddings for feature processing, mapping high-dimensional sparse feature vectors to low-dimensional dense feature vectors. The model incrementally learns recommendation samples using an Adam optimizer and returns difficult-to-recommend samples. The sequential three-way decision strategy is used to increase the delay in decision making, and sequential thresholds are used at different granularity levels to dynamically partition difficult-to-recommend samples [14].

Yan et al. proposed a graph neural network recommendation algorithm fused with semantic information and attention. They first pointed out that as the scale of original data in recommendation systems continues to expand, a significant volume of text data containing semantic information has not been effectively utilized, and the graph neural network merges neighbor information in the graph without distinguishing critical nodes, making it challenging for the model to acquire high-quality entity features, ultimately resulting in a drop in recommendation precision. Then, they considered that users and items themselves have certain attribute information, such as comments and product descriptions, which contain a large amount of semantic information. Meanwhile, different neighbors in the graph have

DOI: 10.54254/2755-2721/14/20230791

different influences on the target node, and they contribute to the embedding representation of the target node with different sizes during the influence propagation process. By using the attention mechanism to calculate the attention weights of neighbor nodes and aggregating the feature information of relevant neighbor nodes according to the weights, the learned embedding representation of the target node can be more accurate and reasonable. By combining the implicit semantic information in this paper with the influence of interaction relationships in the network, the model can learn the embedding representations of items and users, thereby improving the accuracy of recommendation results [15].

3. Methodology

3.1. Collaborative filtering algorithm

Collaborative filtering (CF) is a widely used recommendation algorithm mainly applied in personalized recommendation systems. This algorithm's central premise is to discover other users or items that share comparable behavioral patterns with the target user, based on past interactions. It then uses this data to estimate the target user's interests or preferences. In simple terms, CF is a recommendation algorithm based on similarity that can make recommendations by finding similarities.

CF can be divided into two types: user-based CF and item-based CF. User-based CF is based on user behavior to make recommendations and predicts the target user's preferences by calculating the similarity between users. Item-based CF is based on the similarity between items to make recommendations, and it recommends based on the similarity between items already rated by the user and the item to be recommended. The advantage of CF is that it can provide personalized recommendation results for users without analyzing and modeling the characteristics of items.

3.2. Algorithm process

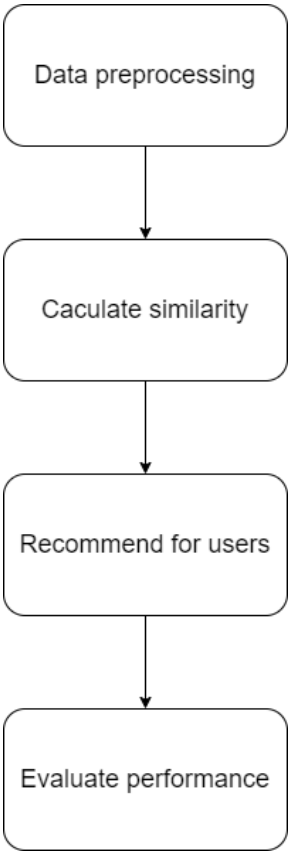

Figure 1. Flowchart of the collaborative filtering algorithm.

DOI: 10.54254/2755-2721/14/20230791

Data preprocessing: Convert each row of the original dataset into a Python data list that can be operated on.

Calculate similarity: Similarity calculation is the core step of the collaborative filtering algorithm. Common similarity calculation methods include the Jaccard formula, cosine similarity, and Pearson correlation coefficient. In this paper, we use a variant of the Jaccard formula as the similarity calculation formula.

Recommend for users: In the user-based collaborative filtering algorithm, we calculate the similarity between the target user and other users and select the k users with the highest similarity as neighbors. In the item-based collaborative filtering algorithm, we calculate the similarity between the target item and other items and select the k items with the highest similarity as the recommendation list. Finally, we recommend the items that neighbors like or the items in the recommendation list to the target user.

Evaluate algorithm performance: to evaluate the accuracy and reliability of the algorithm, we use common evaluation metrics, namely precision, recall, coverage, and popularity, to evaluate the algorithm's performance.

3.3. User-based CF

The main logic of User-CF is to calculate the similarity between each user and all other users who have interaction records with this user, and then find the k most similar users to the target user based on similarity, and recommend the items interacted with by these k users to the target user. Based on this logic, according to the algorithm flow presented in the previous section, we can clearly define the entire process of User-CF:

1) First, the User-Item interaction records are to be read as an operable data list, and the training set and the test set are split according to a certain ratio.

Data is [user1, item1, user2, item2, user3, ……]

2) Next, two data dictionaries are created based on the training set, one to store the items rated by each user and the other to store the users who have rated each item. For the former, the key of the dictionary is the user ID, and the value is a set containing the IDs of all the items rated by this user. For the latter, the key is the item ID, and the value is a set containing the IDs of all the users who have rated this item.

User_item_dict is { user1 : { item2, item4, item6, …}, user2 : {item1, item3, …}, …}

Item_user_dict is {item1 : {user1, user2, user4, …}, item2 : {user3, user4, user5, …},…}

3) Next, build the User-User similarity matrix dictionary, which calculates the similarity between a user and all other users who have interaction records with this user. For each key-value pair in the similarity matrix dictionary, the key is a user, and the value is a dictionary composed of all other users with interaction records and their corresponding similarity.

User_similarity_dict is { user1 : { user1 : similarity1, user2 : similarity2, …} …}

User-CF does not calculate similarity between two users based on their tags or profiles, but rather on the items they both like. User-CF believes that User1 and User2 are highly similar because most of the items that User1 likes are also liked by User2. Therefore, to find the similarity w_{uv} between user u and user v, it is necessary to find how many items in the set of items that user u likes, denoted as $N(u)$, also exist in the set of items that user v likes, denoted as $N(v)$. This can be achieved by calculating the intersection of the two sets and then dividing the result by the geometric mean of the two sets. The formula is as follows:

Similarity:

$$w_{uv} = \frac{|N(u) \cap N(v)|}{\sqrt{|N(u)||N(v)|}}$$

In real life, we may find that if two users both give high ratings to the movie "Star Wars", it does not necessarily reflect their common interests because this is a widely known movie. We do not want highly popular movies to overly influence the similarity calculation between two users. Therefore, we need to use an improved similarity calculation method:

DOI: 10.54254/2755-2721/14/20230791

Advanced Similarity:

$$W_{uv} = \frac{\sum_{i \in N(u) \cap N(v)} \frac{1}{\log(1 + |N(i)|)}}{\sqrt{|N(u)||N(v)|}}$$

The $\frac{1}{\log(1+|N(i)|)}$ in the modification further reduces the impact of highly popular items on similarity calculation [16]. For less popular items, if both users rate them highly, it indicates a strong similarity in their preferences. This can be implemented using the following code:

```
for ui, related_user in self.user_similarity_martix.items():
    for uj in related_user:
        self.user_similarity_martix[ui][uj] /=
        sqrt(len(self.user_item_dct[ui]) * len(self.user_item_dct[uj]))
```

4) Finally, for each user, calculate the recommendation coefficient rec_{ui} for all items with respect to this user and sort all items in descending order according to the recommendation coefficient. Select the top k items with the largest recommendation coefficients for recommendation.

Recommended coefficient:

$$rec_{ui} = \frac{\sum_v W_{uv} r_{vi}}{\sum_v W_{uv}}$$

In this formula, v runs over k users with maximum W_{uv}.

3.4. Item-based CF

The main logic of Item-CF is to first calculate the similarity between items based on the historical preference data of all users, and then find the k most similar items to the target item that the user likes and recommend these k items to the user. Following this logic, we can clearly define the entire process of Item-CF based on the algorithm flow given above:

1) Firstly, the user-item rating records are to be read and converted into a manipulable data list, and then the records are split into training and test sets according to a certain ratio.

Data is [user1, item1, rating1, user2, item2, rating2, user3, ……]

2) Then, based on the training set, two data dictionaries are established: one stores the items that each user has rated, and the other stores the users who have rated each item. For the former, the key of the dictionary is the user ID, and the value is a set of all item IDs that this user has rated. For the latter, the key is the item ID, and the value is a set of all users who have rated this item.

User_item_dict is { user1 : { item2, item4, item6, …}, user2 : {item1, item3, …}, …}

Item_user_dict is {item1 : {user1, user2, user4, …}, item2 : {user3, user4, user5, …},…}

3) Next, to construct the Item-Item similarity matrix dictionary, the algorithm calculates the similarity between each item and all other items. For each key-value pair in the similarity matrix dictionary, the key represents an item and the value represents a dictionary composed of all other items and their corresponding similarity scores.

Item_similarity_dict is { item1 : { item2 : similarity1, item3 : similarity2, …} …}

Item-CF does not conduct the content attributes of items to calculate the similarity between items, but instead analyzes user behavior records to calculate the similarity between items. Item-CF believes that Item1 and Item2 have a high degree of similarity because most of the users who like Item1 also like Item2. Therefore, to find the similarity W_{ij} between items i and j, it is necessary to find how many of the users who like item i in the set $N(i)$ also like item j in the set $N(j)$. This is done by calculating the intersection of the two sets and then dividing it by their union using the geometric mean. The formula is as follows:

Similarity:

DOI: 10.54254/2755-2721/14/20230791

$$W_{ij} = \frac{|N(i) \cap N(j)|}{\sqrt{|N(i)||N(j)|}}$$

Matrix normalization can scale the values in the similarity matrix between 0 and 1 or transform it into a standard normal distribution. This can eliminate the proportional differences that may exist between different items, reduce the influence of outliers, and improve the quality of the similarity matrix. In addition, normalization also helps reduce computational costs because, after normalizing the similarity matrix, calculations can be performed faster, and storage space can be reduced.

If the values in the similarity matrix have the same scale and range, it can significantly improve the accuracy of the prediction results. Therefore, we have improved the above formula by normalizing the calculated similarity matrix.

Advanced Similarity:

$$W'_{ij} = \frac{W_{ij}}{max_j W_{ij}}$$

This can be implemented using the following code:

```
for item1, items in item_similarity_matrix.items():
    ms = max(items.items(), key=lambda k: k[1])[1]
for item2, item2_value in items.items():
    if item2_value != 0:
        item_similarity_matrix2[item1][item2] = item2_value / ms
```

4) Finally, we rank all other items based on the recommendation score calculated for each item, and sort them in descending order. After calculating the recommendation scores, we recommend the k items with the highest scores.

Recommended coefficient:

$$rec_{ui} = \frac{\sum_j W_{ij} r_{uj}}{\sum_j W'_{ij}}$$

In this formula, j runs over k items with maximum W_{ij} .

3.5. The comprehensive comparison between User-CF and Item-CF

3.5.1. Similarity calculation. User-CF calculates similarity based on the similarity between users, using methods such as cosine similarity or the Pearson correlation coefficient.

Item-CF calculates similarity based on the similarity between items, also using methods such as cosine similarity or the Pearson correlation coefficient.

3.5.2. Recommendation generation. User-CF: For a user, first find all the users similar to this user, then calculate the recommendation coefficient of these users for items, and recommend high-coefficient items to the target user.

Item-CF: For a user, based on the items that the user has rated, find items similar to these items, calculate the weighted average score of these similar items, and recommend high-score items to the user.

3.5.3. Advantages and disadvantages.

User-CF:

Advantages:

a) It can provide recommendations for new users or solve cold-start problems.

b) It has good explanatory power for user behavior, as the recommendation results are based on the behavior of similar users.

Disadvantages:

DOI: 10.54254/2755-2721/14/20230791

a) When the number of users is large, a large number of similarity matrices between users need to be calculated, which requires a lot of computation.

b) When the user's behavior changes, the similarity matrix needs to be recalculated.

Item-CF：

Advantages:

a) It can avoid the need to recalculate the similarity matrix when user interests change.

b) It can handle large-scale datasets because it only needs to calculate the similarity matrix between items, not between users.

c) The recommendation results are relatively stable because the characteristics of the items are relatively stable.

Disadvantages:

a) It cannot provide recommendations for new users or solve cold-start problems.

b) When the item inventory changes, the similarity matrix needs to be recalculated, which requires a lot of computation.

In summary, User-CF is suitable for situations where user behavior changes frequently, the total amount of items is greater than the total amount of users, and the computation required is relatively large, such as on news websites or social networking sites. Item-CF is suitable for situations where the total amount of items is relatively stable and the total amount of users is greater than the amount of items, and it requires relatively little computation.

4. Result and discussion

4.1. Experimental purpose and evaluation metrics

To verify the correctness of the User-CF method and the Item-CF method described in this paper, as well as the effectiveness of their improved similarity methods, the author conducted a controlled experiment on the same dataset, measuring the precision, recall, recall at k, coverage, and popularity of the two methods and their improved similarity methods at k (recommendation quantity) of 5, 10, 20, 30, and 50.

Their calculation formulas are shown below:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$recall\ at\ k = \frac{TP\ at\ k}{TP\ at\ k + FN}$$

$$coverage = \frac{n}{N}$$

$$popularity = \frac{i}{N}$$

The TP, FP, TN, and FN in the formulas respectively represent True Positive, False Positive, True Negative, and False Negative, and their definitions are as follows [17]:

1) TP (True Positive): The quantity of recommendations in the recommendation result set that exist in the test set.

2) TP at k (True Positive at k): The quantity of recommendations in the top k recommendation result set that exist in the test set.

3) FP (False Positive): The quantity of recommendations in the recommendation result set that do not exist in the test set.

4) TN (True Negative): The data that neither exists in the test set nor in the recommendation result set.

DOI: 10.54254/2755-2721/14/20230791

5) FN (False Negative): The data that exists in the test set but not in the recommendation result set.

In addition, n is the quantity of item categories in the recommendation result, N is the quantity of item categories in the item pool, and i represents the quantity of times a certain item is interacted with by users.

4.2. Experimental environment

The environment of the computer used by the author for this experiment is as follows:

Table 1. Computing environment used by the author.

CPU	Intel(R) Core(TM) i9-12950HX
GPU	NVIDIA GeForce RTX 3070 Ti Laptop
Memory	32GB 4800Mhz DDR5
Operating System	Windows 11 21H2

4.3. Experimental dataset

This author selects the classic movie dataset MovieLens-latest-small (referred to as ML-small for short) in the field of recommendation systems. The ML-small dataset describes 5-star ratings and user-generated free-text tags from MovieLens, and to ensure the universality of the dataset, ML-small randomly selects users, each of whom has rated at least 20 movies. Users are distinguished from each other only by different IDs, and no other information is included.

The statistics of the ML-small dataset are represented in the following table:

Table 2. Statistics of the ML-small dataset.

Dataset	Users	Movies	Records	Sparsity
ML-small	610	193609	100836	99.91%

ML-small contains four table files: links.csv, movies.csv, ratings.csv, and tags.csv. The User-CF and Item-CF methods described in this article only require the User-Item rating matrix to run, so we only need to use the movies.csv file.

4.4. Dataset split

In this study, the training set and test set are divided at a ratio of 3:1. The training set is used to run the algorithm models and obtain recommendation results, while the test set is used to measure the five evaluation indicators of precision, recall, recall at k, coverage, and popularity of the recommendation results. The training set and test set must be randomly partitioned; otherwise, it will seriously affect the accuracy of the experimental results.

5. Experimental results and analysis

User-CF. Calculation results based on similarity before improvement:

Table 3. Performance of User-CF before similarity improvement.

K	Precision	Recall	Recall at k	Coverage	Popularity
5	0.3082	0.0376	0.3110	0.0089	5.2741
10	0.2643	0.0636	0.2799	0.0129	5.1540
20	0.2180	0.1054	0.2826	0.0232	5.0070
30	0.1803	0.1311	0.2862	0.0308	4.9483
50	0.1515	0.1826	0.3317	0.0424	4.8123

Calculation results based on improved similarity:

DOI: 10.54254/2755-2721/14/20230791

Table 4. Performance of User-CF after similarity improvement.

K	Precision	Recall	Recall at k	Coverage	Popularity
5	0.3013	0.0367	0.3029	0.0095	5.2579
10	0.2708	0.0652	0.2912	0.0151	5.1255
20	0.2165	0.1043	0.2822	0.0226	5.0228
30	0.1874	0.1354	0.2952	0.0316	4.9320
50	0.1578	0.1909	0.3330	0.0449	4.7982

Observing the table, we can make the following findings:

For both tables, as the value of k increases, precision and popularity gradually decrease. This is because the increase in k leads to an increase in the amount of user neighbors and the amount of recommended items, which may include items recommended by dissimilar users, resulting in a decrease in precision. Additionally, the number of interactions between items and users decreases, resulting in a decrease in popularity.

For both tables, as the value of k increases, recall, recall at k, and coverage gradually increase. This is because as the k value increases, the size of the recommendation result set increases, making it more likely to contain items that the user is interested in, resulting in an increase in recall. At the same time, the set of recommended items will cover more item types, resulting in an increase in coverage.

Using the improved similarity calculation of User-CF compared to the original version, while maintaining precision and recall relatively stable, its coverage has increased by an average of 5.94% for each different k value, and its popularity has decreased by an average of 0.23%. This indicates that the improved similarity calculation reduces the influence of highly popular items on recommendation results, allowing the algorithm to more comprehensively discover users' low-popularity interests and provide recommendation results with higher coverage and lower popularity compared to before.

The algorithm performs well when $k<10$, with the highest precision reaching nearly 30%, which can provide relatively accurate recommendations for users. However, when $k\geq10$, the precision drops dramatically, with the precision at k=50 being only half that at k=5, making it difficult to provide precise recommendations for users.

Item-CF. Calculation results based on similarity before improvement:

Table 5. Performance of Item-CF before similarity improvement.

K	Precision	Recall	Recall at k	Coverage	Popularity
5	0.2915	0.0349	0.2941	0.0475	4.5160
10	0.2505	0.0598	0.2713	0.0763	4.4394
20	0.2078	0.1019	0.2800	0.1048	4.4671
30	0.1832	0.1327	0.3009	0.1412	4.3642
50	0.1516	0.1835	0.3376	0.1857	4.3210

Calculation results based on improved similarity:

Table 6. Performance of Item-CF after similarity improvement.

K	Precision	Recall	Recall at k	Coverage	Popularity
5	0.2852	0.0345	0.2899	0.0485	4.5288
10	0.2454	0.0586	0.2675	0.0698	4.4852
20	0.2243	0.1087	0.2960	0.1006	4.4736
30	0.1834	0.1326	0.3000	0.1229	4.4133
50	0.1554	0.1868	0.3524	0.1640	4.3281

Observing the table, we can find the following patterns:

A. For two tables, as the value of k increases, precision and popularity gradually decrease, while

DOI: 10.54254/2755-2721/14/20230791

recall, recall at k, and coverage gradually increase. The specific reason has been explained in the previous paragraph.

B. When $k \leq 10$, improving the similarity measure cannot bring performance improvements to the algorithm. However, when $k>10$, using the improved similarity measure for the Item-CF algorithm can slightly improve precision, recall, and recall at k, while other metrics remain basically the same or even slightly lower than the original Item-CF algorithm. This indicates that normalizing the item similarity matrix does not necessarily significantly improve the performance metrics of the Item-CF algorithm. Normalization is aimed at eliminating the difference in similarity measurement scales between different items so that the numerical range of the similarity matrix is between 0 and 1, making it easier to compare and process. Moreover, due to the small size of the ML-small dataset used in this paper, the inherent inaccuracy of the item similarity measurement method, the long-tail effect of items in Item-CF, user interest drift, and other reasons, we cannot see the performance improvement brought by normalization.

C. The algorithm's precision can reach up to 30% when $k<10$, which can provide users with relatively accurate recommendations.

D. The coverage of Item-CF is much higher than that of User-CF. Compared with their improved methods, the coverage of Item-CF is on average 3.344 times that of User-CF. This is because the Item-CF algorithm itself uses the similarity between items, making it easier to discover and recommend popular items. User-CF is more dependent on the similarity between users, so it may be more likely to ignore niche items. For example, if there is a group of users who have similar interests in a niche item, but their similarity is low, User-CF may ignore this item, while Item-CF is more likely to discover this correlation and recommend this item to the target user, thereby improving coverage.

6. Conclusion

In the context of the internet business era, excellent recommendation systems can save users decision-making time and improve their software usage experience, as well as bring more revenue to merchants or enterprises in the same amount of time. Therefore, the core of the recommendation system is to accurately discover users' interests and provide personalized recommendations for them.

This article discusses the classic collaborative filtering recommendation algorithm in the recommendation system, which is divided into User-CF and Item-CF, respectively, explaining their algorithm principles, algorithm processes, data formats, similarity calculation formulas and their improvements, recommendation principles, and pros and cons comparison. In User-CF, the algorithm finds similarities between users and recommends products to the target user based on those similar users. In Item-CF, the algorithm recommends items similar to those liked by the target user by calculating the similarity between items. The core idea of these methods is to analyze users' interests through their historical behavior, predict items they may be interested in, and recommend them to the target user.

The author studied five performance indicators of User-CF and Item-CF: precision, recall, recall at k, coverage, and popularity. The author found in the experiment that Item-CF has better coverage and popularity in recommendations, while User-CF is more suitable for improving precision and recall in recommendations. The author also found that normalizing the similarity matrix can slightly improve the algorithm's performance, but normalization has little effect on coverage and popularity.

Due to hardware limitations, dataset size limitations, and the robustness of similarity calculations, the author's research cannot significantly improve the algorithm for all indicators. Also, after increasing k, the decrease in precision makes both User-CF and Item-CF unable to provide accurate recommendations for small k.

In the future, we can use methods such as cosine similarity weighting, fusing latent semantic models, fusing item content with knowledge graphs, and fusing time factors to improve Item-CF's similarity calculation. We can also use artificial neural networks, matrix factorization, graph path random walks, graph convolutional neural networks, and other methods to improve User-CF's similarity calculation.

In practical applications, collaborative filtering algorithms have been widely used in many fields, such as social networks, e-commerce, music, and movie recommendations. Although collaborative filtering algorithms have some advantages, such as easy implementation and strong scalability, they also

have some disadvantages, such as cold start problems, sparsity problems, and gray group problems. Therefore, we need to continue to research and optimize collaborative filtering algorithms to better deal with the problems encountered in practical applications. In the future, we believe that collaborative filtering algorithms will continue to play a significant role in the field of recommendation systems.

References

[1] ZHANG Qi, YU Shuang-yuan, YIN Hong-feng and XU Bao-min. (2023). Neural Collaborative Filtering for Social Recommendation Algorithm. Computer Science (02),115-122.

[2] XIA Hongbin ,LU Wei ,LIU Yuan.(2021). Collaborative Filtering with Heterogeneous Neighborhood Aggregation. Pattern Recognition and Artificial Intelligence (08),712-722.

[3] LIU Guo-zhen & CHEN Hong-long. (2021). Convolutional Memory Graph Collaborative Filtering. Journal of Beijing University of Posts and Telecommunications (03),21-26.

[4] LIU Rui-hong, ZHOU Xu-jie, YUAN Guang-hui & LIU Zhao-chun. (2022). An exercise recommendation algorithm integrating collaborative filtering and classification. Computer knowledge and Technology (28),23-25.

[5] JIANG Zong-Li, TIAN Cong-Cong .(2021). Collaborative Filtering Algorithm of Graph Neural Network Based on Fusion Meta-Path. Computer Systems & Applications (02),140-146.

[6] SU Jing, XU Tian-qi, ZHANG Xian-kun, SHI Yan-cui & GU Shu-ting. (2021).Collaborative filtering recommendation model based on graph convolution and cross product. Application Research of Computers (10),3044-3048. doi: 10.19734/j.issn.1001-3695.2021.02.0053.

[7] SONG Yong-lei & CAO Jian-guo. (2021). Graph Neural Collaborative Filtering Model Research Based on Self-Attention. Journal of Southwest China Normal University (Natural Science Edition) (07),130-134. doi: 10.13718/j.cnki.xsxb.2021.07.019.

[8] GAO Fei, LIN Kai-jie. (2021). Bidirectional Collaborative Filtering Recommendation Algorithm Based on Graph Convolution Network. SOFTWARE (07),32-38.

[9] YAO Nan, WANG Hong-sheng. (2021). Research on Recommendation Algorithm based on Graph Neural Network. Changjiang Information & Communications (10),41-43.

[10] CHEN Yi-fan, ZHU Min-yao, ZHU Xiao-qiang, SONG Hai-yang & LU Xiao-feng. (2022). Graph convolution collaborative filtering recommendation algorithm based on the time series features. ELECTRONIC MEASUREMENT TECHNOLOGY (06),79-85. doi: 10.19651/j.cnki.emt.2108462.

[11] LI Lin, WANG Pei-pei, DU Jia & ZHOU Dong. (2022). Group Recommendation Method with Nash Equilibrium Strategy and Neural Collaborative Filtering. Pattern Recognition and Artificial Intelligence (05),412-421.

[12] CHEN Hui, Wang Kai-yue. (2022). Deep neural network matrix factorization recommendation algorithm based on similarity calculation. Journal of Shandong University of Technology (Natural Science Edition) (06),67-73. doi: 10.13367/j.cnki.sdgc.2022.06.009.

[13] Zhang Xian-kun, Qin Feng-bin, Sun Yue, Huang Wen-jie, Dong Mei. (2023). Neural graph collaborative filtering based on interaction intent. Application Research of Computers (02),488-492. doi: 10.19734/j.issn.1001-3695.2022.07.0354.

[14] CHENG Shu-hui, WU You-xi. (2023). Personalized recommendation based on sequential three-way decisions with single feedforward neural network. Control and Decision. doi: 10.13195/j.kzyjc.2022.1227.

[15] YAN Yang, WANG Lei-Quan, LI Jia-Rui. (2022). Graph Neural Network Recommendation Algorithm Fused with Semantic Information and Attention. Computer Systems & Applications. doi: 10.15888/j.cnki.csa.009016.

[16] LI Mei. (2022). Research on News Recommendation Algorithm Based on Collaborative Filtering. Computer Knowledge and Technology (34),51-53. doi: 10.14004/j.cnki.ckt.2022.2228.

[17] Tom Fawcett. An introduction to ROC analysis. Pattern Recognit. Lett. 27(8): 861-874 (2006)