# Improvement and Implementation of a Spark-Based Hybrid Collaborative Filtering Algorithm (Postprint)

**Authors:** Wang Yuanlong, Sun Weizhen, Xiang Yong

**Date:** 2018-05-18T00:00:00+00:00

## Abstract

To address the sparsity, scalability, and personalization issues inherent in traditional collaborative filtering recommendation processes, this work introduces the ensemble learning concept to optimize and improve a novel hybrid collaborative filtering framework on the Spark platform. Drawing upon the Stacking ensemble learning paradigm, multiple weak recommenders are combined through linear weighting to form a comprehensive and robust recommender. First, the algorithm builds upon neighbor-based collaborative filtering, optimizing the neighbor similarity computation strategy by incorporating classification, popularity, and rating quality metrics. This aims to enhance the rationality of similarity measures while reducing computational complexity, thereby alleviating the rating sparsity problem to a certain extent. Simultaneously, the algorithm leverages the Spark distributed computing platform, capitalizing on its advantages in stream processing and distributed storage architectures to design and implement an incremental iterative model for the recommendation algorithm, which resolves the scalability and real-time performance issues of collaborative filtering. Experimental evaluations are conducted using the publicly available UCI MovieLens dataset and Netflix movie rating data. The results demonstrate that the improved algorithm exhibits promising performance in recommendation personalization, accuracy, and scalability, achieving varying degrees of improvement over previous comparable algorithms, thus providing a feasible algorithmic integration solution for recommendation system applications.

## Full Text

## Preamble

**New Improvement and Implementation of Hybrid Collaborative Filtering Algorithm Based on Spark Platform**

Wang Yuanlong[1], Sun Weizhen[1], Xiang Yong[2]
(1. Department of Computer Science and Technology, College of Information Engineering, Capital Normal University, Beijing 100048, China;
2. Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

**Abstract:** To address the sparsity, scalability, and personalization issues inherent in traditional collaborative filtering recommendation processes, this paper introduces an algorithm integration approach to optimize and improve a novel hybrid collaborative filtering algorithm on the Spark platform. Drawing on Stacking ensemble learning principles, multiple weak recommenders are linearly weighted and combined to form a comprehensive recommender. First, the algorithm optimizes the nearest-neighbor similarity computation strategy by incorporating classification, popularity, and rating quality metrics, thereby improving the rationality of similarity measures and reducing computational complexity while mitigating rating sparsity to some extent. Simultaneously, by leveraging the Spark distributed computing platform' s advantages in stream processing and distributed storage, the algorithm designs and implements an incremental iterative model for recommendation, solving the scalability and real-time performance problems of collaborative filtering. Experiments conducted on the public UCI MovieLens and Netflix rating datasets demonstrate that the improved algorithm achieves favorable performance in recommendation personalization, accuracy, and scalability, showing varying degrees of improvement over previous similar algorithms and providing a feasible algorithm integration scheme for recommendation system applications.

**Keywords:** ensemble learning; collaborative filtering; sparsity; scalability; Spark streaming; incremental model; classification

**Funding:** Beijing Municipal Education Commission Science and Technology Plan Project (KM201310028014)

**Author Biographies:** Wang Yuanlong (1988-), male, master' s student, research interests include data mining and big data processing; Sun Weizhen (1963-), male, associate researcher, research interests include operating systems, image processing, and data mining; Xiang Yong, associate professor, Ph.D., research interests include computer-supported cooperative work, operating systems, computer networks, and data mining.

## 0 Introduction

In the era of big data where the Internet and IoT intersect, information overload has become a severe challenge for users. Faced with massive amounts of data, accurately and efficiently finding needed information has become increasingly difficult. Recommendation systems represent an effective solution to this problem. A recommendation system is an algorithmic framework de-

signed to provide information recommendations, reduce the difficulty for users to find effective information, and enhance the interactive experience between users and network information. The core of recommendation systems lies in recommendation algorithms, which have deep connections with cognitive science, information retrieval, mathematics, social sciences, and other fields, making it a multidisciplinary research direction.

Recommendation systems originated in the 1990s and have undergone more than two decades of development. Their theoretical foundations continue to evolve and improve, with applications becoming increasingly widespread. In today' s Internet-IoT convergence era, major e-commerce platforms, search engines, news portals, and social networking sites—such as Amazon, JD.com, Yahoo, Weibo, and Netflix—all incorporate recommendation systems. Furthermore, recommendation systems are increasingly being applied in other domains including intelligent transportation, smart homes, and artificial intelligence. Consequently, research on recommendation systems has become both urgent and meaningful.

Traditional recommendation methods typically fall into two categories: collaborative filtering (CF) and content-based methods. Collaborative filtering is the focus of this research. Content-based recommendation utilizes fine-grained user historical data, where the recommended information is often based on content itself, such as news or web documents. While this approach yields relatively accurate recommendations, it suffers from content limitations, fails to provide novel recommendations, and does not protect user privacy effectively.

Collaborative filtering algorithms can be divided into two types: nearest-neighbor based and model-based. The fundamental principle of collaborative filtering is collective intelligence learning—predicting a user' s behavior through their friends' behaviors. The greatest advantage of collaborative filtering is its ability to discover items that users might potentially be interested in, making recommendations novel while protecting user privacy. Nearest-neighbor collaborative filtering can be user-based or item-based. User-based collaborative filtering predicts user preferences by analyzing user ratings, while item-based collaborative filtering makes recommendations by analyzing item similarity. Model-based approaches attempt to further fill the rating matrix by employing machine learning algorithms to train rating vectors and build models for predicting user ratings on new items.

Collaborative filtering faces two major challenges: how to address rating sparsity and algorithm scalability. Previous researchers have proposed various improved collaborative filtering algorithms. For instance, some have enhanced recommendation efficiency and personalization for user-based collaborative filtering, achieving good personalization but failing to address the critical issue of scalability. Matrix factorization-based collaborative filtering, typified by SVD algorithms, offers an effective means to solve sparse matrix problems with the advantage of speed, yet it also fails to resolve scalability issues. Content-based recommendation algorithms perform well in scalability but often lack recommendation novelty, diminishing their value. Spark-based collaborative filtering

recommendation algorithms leverage the massive data processing capabilities of distributed platforms but offer limited algorithmic improvements. Currently, the only Spark implementation for recommendation algorithms is based on least squares, which demonstrates inadequate performance in scalability and real-time processing.

Therefore, the objective sought in this paper is to achieve an optimal balance among personalization, novelty, and scalability, striving for breakthroughs in one or two aspects compared to previous algorithms. This algorithm has three innovative points: First, the implementation platform is based on Spark' s distributed platform, which differs from standalone models. Leveraging Spark clusters significantly enhances algorithm parallelism. Second, this algorithm successfully integrates the advantages of user-based, item-based, and model-based recommendation algorithms to achieve better recommendation personalization. To reduce implementation complexity, the algorithm focuses on improving collaborative filtering, predicting only user behavior and rating data without involving content-based recommendations. Third, the algorithm implements an incremental model that enables iterative computation for real-time recommendation, representing a major highlight. Due to the fusion of multiple algorithmic ideas, to reduce implementation difficulty, the decision was made not to incorporate content-based recommendation algorithms. In summary, this algorithm considers the combination of user-based, item-based, and model-based approaches, using pre-classification to reduce similarity computation, employing popularity and rating quality metrics to adjust weights for user and item similarities, obtaining user-based and item-based recommendation lists, and finally performing weighted fusion with least squares recommendation. This model resembles a random forest approach, and based on the principle of combining weak classifiers linearly to form a strong classifier, the final model represents an enhanced hybrid recommendation model.

---

## 1 Related Work

Traditional recommendation methods are typically divided into two major categories: collaborative filtering (CF) and content-based methods, with collaborative filtering being the research focus of this paper.

Content-based recommendation is a fine-grained approach that utilizes user historical data records, where the recommended information carrier is often based on the content itself, such as news articles or web documents. Its advantages include relatively accurate recommendations, but its disadvantages are that recommendations are limited by content, cannot achieve novelty, and do not protect privacy effectively.

Collaborative filtering algorithms can be categorized into nearest-neighbor based and model-based approaches. The fundamental idea of collaborative filtering is the principle of collective intelligence learning—predicting a user' s behavior

through their friends' actions. The greatest advantage of collaborative filtering is its ability to uncover items that users may potentially be interested in, making recommendations novel while protecting user privacy.

Nearest-neighbor collaborative filtering can be specifically divided into user-based and item-based methods. User-based collaborative filtering predicts user preferences by analyzing user ratings, while item-based collaborative filtering makes recommendations by analyzing item similarity. Item-based recommendations utilize inter-item similarity to generate recommendations for users. Model-based approaches attempt to further fill the rating matrix by employing machine learning algorithms to train rating vectors and build models for predicting user ratings on new items.

Collaborative filtering algorithms face two major problems: how to solve rating sparsity and algorithm scalability. This research will address these two issues from the perspective of distributed computing. Previous researchers have proposed some improved collaborative filtering algorithms, such as those focusing on recommendation efficiency and personalization for user-based collaborative filtering, which achieved good personalization but failed to solve the most critical problem of scalability. Matrix factorization-based collaborative filtering, typified by SVD algorithms, offers an effective solution for sparse matrices with the primary advantage of speed, but similarly fails to address scalability issues. Content-based recommendation algorithms perform well in scalability but often make recommendations lack novelty, reducing their significance and value. Spark-based collaborative filtering recommendation algorithms leverage the massive data computing capabilities of distributed platforms but do not offer substantial algorithmic improvements. Currently, the only Spark implementation for recommendation algorithms is based on least squares, which demonstrates poor performance in scalability and real-time processing.

---

## 2 Detailed Design of the Improved Algorithm

### 2.1 Algorithm Improvement Ideas

This paper primarily employs algorithm integration ideas to reasonably optimize and combine multiple collaborative filtering algorithms, leveraging their respective strengths to maximize algorithm efficiency and prediction accuracy. In general, the algorithm performs the following tasks during implementation: pre-classification of data objects, elimination of unnecessary similarity computations, improvement of similarity rationality, utilization of stream computing to implement incremental models, and integration of algorithms with platforms.

At the platform level, the Spark distributed computing platform is adopted. By leveraging the high reliability and high performance characteristics of distributed systems, the platform not only provides powerful computational support for algorithm execution but, more importantly, utilizes Spark' s stream

processing model to complete the computation of algorithm incremental models, addressing the scalability issues inherent in collaborative filtering algorithms.

At the algorithmic level, pre-classification techniques are first used to solve the problem of numerous unnecessary similarity computations and storage challenges caused by rating sparsity. Specifically, before computing similarity, items and users are classified according to tags, which are saved for rapid category judgment of incremental data, enabling fast pattern matching and completing fuzzy recommendations. Next, for already grouped users, methods such as building inverted indexes are employed within groups to further reduce pairwise similarity computation frequency and perform original similarity calculations. After completing original similarity computation, popularity and rating quality metrics are used to adjust the previously calculated user similarity. The rule is: reduce similarity between users corresponding to highly popular items, and increase similarity between users corresponding to highly-rated items. After obtaining user similarity, appropriate nearest neighbors are determined based on user similarity, and recommendation item ratings are calculated for each user according to their similarity with neighbors and their ratings, forming recommendation lists. Then, item-based collaborative filtering is applied to obtain another recommendation list, and similarly, matrix factorization methods are used to calculate a third recommendation list. Finally, linear weighted averaging is performed to obtain the final recommendation list. The rationale is that if an item appears in all three recommendation lists, its final calculated score will be relatively high; if it appears in only one list, the recommendation credibility is low and the score will obviously decrease. This not only provides good explainability for recommendations but also improves recommendation quality. After model establishment, cross-validation is used for model evaluation and selection.

### 2.2 Runtime Process of the Improved Recommendation Algorithm

The stream computing process of the recommendation algorithm based on the Spark platform can be described as follows: After the Spark distributed program starts, it first loads the driver through the context, acquires and allocates computing resources, and launches daemon processes. Then, Spark Streaming reads real-time data from the Kafka message queue, performs real-time data classification, and computes user similarity and item similarity in real-time to obtain recommendation lists. The recommendation list from least squares matrix factorization is weighted and merged with the previously obtained recommendation lists to produce and update the final recommendation list and its predicted scores. This process is illustrated in [Figure 1: see original paper].

#### 2.3.3 User and Item Pre-classification

For collaborative filtering algorithms, which represent a coarse-grained recommendation approach for predicting user behavior, the characteristic data primarily includes user ratings, user attributes, item attributes, user click-through rates, dwell time, etc. This paper uses the MovieLens and Netflix movie rating

datasets as the main input feature data sources. The typical rating data format is "userID:movieID:rating:timestamp," whose disadvantage is that a large amount of information is hidden and requires deep mining, and the data is concentrated together, making relationships difficult to discern. Therefore, before computing similarity, user and item attributes are first considered for simple classification of different users and items. This measure yields multiple benefits: it reduces similarity computation complexity, alleviates the sparsity problem of large rating matrices, and facilitates coarse-grained real-time recommendations—particularly important for solving the cold-start problem in collaborative filtering.

The user classification process is as follows: An inverted index is built from the "user-item-rating" table to classify users. As shown in [Figure 4: see original paper], the basic classification idea is to group users or items with the same rating items into one category. Users or items without rating items, referred to as isolated items, are grouped into one category for subsequent algorithmic recommendations. Note that when classifying user or item categories, they need to be numbered with substantive meaning, and certain relationships must be established between different numbers of users and items to facilitate rapid real-time recommendations. The item classification process follows the same principle as users. [Figure 5: see original paper] illustrates the classification process for both users and items.

### 2.3.1 Platform Transition from Standalone to Cluster

The basic idea of transitioning from standalone to cluster involves using open-source distributed frameworks to logically connect inexpensive PCs into a logical whole with a master-slave control structure. The master node performs task scheduling, distribution, and fault tolerance for slave nodes, while slave nodes implement parallel computing—a structure proven to provide highly reliable, highly concurrent, and high-performance computing capabilities. The experimental cluster environment consists of five physical machines. The cluster physical structure is shown in [Figure 2: see original paper]-a, and the cluster logical structure is shown in [Figure 2: see original paper]-b.

### 2.3.2 From Offline to Online Recommendation

Traditional recommendations often involve offline algorithms, but this paper implements both online and offline computation. Leveraging the second-generation distributed data processing architecture Spark, with Kafka as the message queue for data sourcing, Spark Streaming is used for stream processing to implement incremental computation models. The offline component primarily utilizes RDD, Spark SQL, and Spark MLlib for efficient matrix operations and machine learning modeling. HDFS serves as the distributed file system for persistent storage, and HBase as the storage database. HDFS provides the carrier for cluster local data persistence. [Figure 3: see original paper] shows the flow structure diagram of real-time and offline computation.

### 2.3.4 Similarity Computation Incremental Model

For user-based collaborative filtering, beyond the sparsity problem, the biggest bottleneck lies in the massive similarity computations required for online recommendations. Recomputing all similarities for each incremental update incurs excessive computational overhead, necessitating adjustments to the similarity computation strategy. The goal is to minimize repeated similarity computations while ensuring minimal impact on similarity accuracy. The same applies to item-based collaborative filtering. Therefore, this algorithm makes the following adjustments to the similarity computation strategy:

a) **Ignoring the impact of incremental data on original object similarities.** Since incremental data is relatively small compared to original data, its impact on original object similarities is generally minor. Therefore, when computing similarities, original object similarities should not be recomputed; directly reading persisted original object similarities is the most effective approach. For new data, what needs to be computed is the similarity between new objects and between new and original objects. The specific strategy is: for incremental input, judge the incremental ID; if the object already exists in the system, it does not participate in this similarity computation, is persisted to the database, and its occurrence count is incremented. When the occurrence count reaches a certain threshold, relevant data is read and participates in similarity computation.

b) **Parallelization of similarity computation.** This parallelization is made possible by pre-classification of data objects. Each cluster has strong internal connections while inter-cluster connections are loose, minimizing mutual constraints and creating a data format highly suitable for distributed parallel computing, enabling parallel computation across all clusters.

c) **Similarity update and persistence.** After incremental similarity computation, similarities are continuously merged with previously existing similarities and persisted to the database, updating if identical or adding if new. Persisted content includes metadata, computed similarities, and some intermediate state variables (similarity update time, unique identifier, recommendation list update time, and corresponding unique identifier), etc. [Figure 6: see original paper] illustrates the similarity and recommendation model update process.

d) **Simplified analysis of similarity computation frequency.** Let the number of existing users in the original database be $m$ and the number of incremental users be $n$. The number of pairwise similarity computations for full combination is *S count*, as shown in Equation (1). The number of similarity computations after simplification is *S count*, as shown in Equation (2):

项n1)...2()1-()...1()(1　　nnnmnmnmCSnnmcount　　项222)1(2　　nnmnCmnSncount

From Equations (1) and (2), it can be inferred that when $m \gg n$, the time complexities are $O(m^2)$ and $O(m)$ respectively, thus $S\ count \ll S\ count$. This means that even when $m$ is large, the number of improved similarity computations will be far smaller than before. Although this approach sacrifices a small amount of similarity accuracy, it achieves a reasonable balance between precision and time complexity, satisfying the data volume requirements of the incremental computation model.

### 2.3.5 Similarity Computation Methods

For nearest-neighbor based collaborative filtering, recommendations are typically made by computing similarity between objects and using highly similar items or users for recommendation. Various similarity measurement methods exist, and different methods significantly impact recommendation results. For user similarity, Pearson similarity is currently considered better than cosine similarity because it includes normalization, reducing similarity bias caused by different user rating habits. For item similarity, cosine similarity is generally adopted. This paper draws on both similarity computation methods and introduces popularity and rating quality factors to adjust weights for certain similarities, aiming to more accurately measure similarity degrees. The similarity formulas are as follows:

Equation (3) is the original cosine similarity (positive value) for computing item similarity. Equation (4) is Pearson correlation for computing user similarity, where $x$ and $y$ represent user IDs. Equations (5) and (6) represent adjusted user similarity and item similarity respectively, where $P$ and $Q$ denote popularity and rating quality (ranging from -1 to 1), with positive values indicating similarity compensation and negative values indicating similarity penalty; and are popularity threshold coefficient and rating quality parameter (positive values), respectively.

yxyxyxyxyxCosiiiiiii ,),(22yyxxyyxxyyxxyyxxyxPearsoniiii ,)()())(),(22)(,(,),(),

### 2.3.6 Rating Prediction

Since user-based collaborative filtering can effectively discover users' potential interests, item-based collaborative filtering can update recommendation results in real-time based on user behavior, and least squares matrix factorization can better handle rating matrix sparsity issues—while least squares has always been a baseline for rating accuracy—this paper fuses least squares in addition to the previous two combinations to fully leverage their respective advantages. The final item rating result is obtained through linear weighted summation of three parts: predicted ratings based on user similarity, predicted ratings based on item similarity, and ratings from least squares matrix factorization.

The general rule for predicting ratings in each individual recommendation algorithm is to multiply similarity by the ratings of similar objects (users or items) and divide by the sum of similarities.

Let the predicted rating of user $u$ on item $i$ be $v$. The prediction algorithm is described as follows:

**Computing user-based collaborative filtering recommendation list:** a) Find $k$ nearest neighbors of user $u$. b) Identify all real ratings for item $i$ from these $k$ nearest neighbors, recorded as Rating($x$,$i$,$v$), where $x$ represents one of the $k$ nearest neighbors of $u$ who has rated $i$. c) Multiply similarity by real ratings, sum them, and divide by total similarity. The predicted rating of user $u$ on item $i$ is given by Equation (7):

```
)(,),(),,(),(),,(PredictusersSetxuxSimivixRatinguxSimiviuusersusersuser
```

**Computing item-based collaborative filtering recommendation list:** a) Find the $k$ highest-rated items by user $u$. b) Identify $n$ items most similar to these $k$ items. c) Multiply similarity by real ratings, sum them, and divide by total similarity to obtain user $u$' s rating for item $i$, as shown in Equation (8):

```
)(,),(),,(),(),,(PredictitemsSetxuxSimiivixRatinguxSimiviutemitemitem
```

**Least squares matrix factorization** minimizes the loss function through iterative solving of optimal latent feature matrices $U$ and $V$ to obtain user $u$' s rating for item $i$. Equations (9) and (10) show the loss function and predicted rating expression for least squares, respectively.

```
222,),(minjmjiuiTjivnunvurVUf )(),(,),,(PritemsSetiusersSetuVUviuedictiuals
```

**Final recommendation result:** The predicted rating Predict($u$,$i$,$v$) equals the weighted average of the three predicted ratings from user-based, item-based, and least squares methods. The rating expression is given by Equation (11), where , , and are the respective recommendation weights.

```
)(),(),(,,1),,(Pr),,(Predict),,(Predict),,(PredictprefSetvitemsSetiusersSetxuviuedictviuviu
```

**Rating update:** Similar to similarity updates, before recommendation updates, the recommendation rating list must be persisted. The principle for updating rating lists is to recalculate, update, and persist recommendation lists only for object clusters whose similarities have changed within the time interval between two updates. For recommendation lists where similarity remains unchanged, relevant records are directly read from the database for recommendation.

---

## 3 Experimental Results

### 3.1 Test Datasets

The experiments adopt the public UCI MovieLens dataset and Netflix dataset, conducting four groups of experiments on 100K (100,000 entries), 1M (approximately 1 million entries), 10M (approximately 10 million entries), and 100M (approximately 100 million entries) datasets. The rating range is 1-5, with

higher values indicating better evaluations. The primary rating data table used follows the format shown in .

**TABLE:1** Data Format

| UserID | MovieID | Rating | Timestamp |
|---|---|---|---|
| User ID | Movie ID | Rating | Timestamp |

### 3.2 Parameter Setting and Debugging

All algorithm parameter debugging and configuration in this experiment are conducted solely on the MovieLens and Netflix movie rating datasets, with no guarantee of similar results on other datasets. The algorithm parameters include classification quantity, user similarity threshold, item similarity threshold, popularity threshold, rating quality, nearest-neighbor threshold, and error recommendation threshold. provides parameter descriptions for nearest-neighbor recommendation and least squares recommendation parameters.

**TABLE:2** Algorithm Parameter Descriptions

### 3.3 Result Evaluation

**1) Root Mean Square Error (RMSE)**
For rating prediction accuracy, RMSE is commonly used for evaluation. Smaller error indicates smaller deviation between predicted and actual values, meaning higher accuracy. The formula is:

```
TrrRMSEuiuiTiu  2,^
```

where $r\_ui$ represents the actual rating of user $u$ on movie $i$, *î_ui* is the predicted rating from the recommendation algorithm, and $|T|$ is the total number of test sets.

**2) Precision, Recall, and F1**
Precision and recall are generally used to evaluate recommendation quality. Precision represents the proportion of items users are truly interested in among all recommended items. Recall refers to the proportion of recommended items that users are interested in among all items users are interested in. Precision and recall generally show a negative correlation trend; therefore, their harmonic mean F1 can more comprehensively measure overall recommendation quality. The formulas for precision, recall, and harmonic mean are:

```
meanmeanmeanmeanUiiiimeanUiiiimeanRPRPFUCAARUBAAP      21, ,
```

where $A\_i$ represents items recommended to a user that the user is interested in, $B\_i$ represents items recommended but not interesting to the user, $C\_i$ represents items the user is interested in but not recommended, $|U|$ is the number of items participating in evaluation, $P\_mean$ represents average precision,

*R_mean* represents average recall, and F1 is the harmonic mean of precision and recall. A larger F1 value indicates higher precision and recall.

### 3.4 Experimental Results and Analysis

In this paper, the improved algorithm is temporarily named **HybirdCF**. The experimental and test results are as follows. This experiment evaluates the algorithm from several dimensions:

**a) RMSE of various algorithms.** Under optimal parameters for each algorithm in the experimental environment, the RMSE of predicted ratings for the improved hybrid collaborative filtering algorithm and several classical collaborative filtering algorithms is shown in [Figure 7: see original paper].

The results demonstrate that under equivalent conditions, the improved algorithm achieves smaller RMSE than pure user-based collaborative filtering, pure item-based collaborative filtering, and even the least squares method built into Spark. The predictions are more accurate, indicating improved prediction accuracy.

**b) RMSE across different data scales.** [Figure 8: see original paper] shows the measured RMSE of the algorithm under optimal parameters across different data scales.

The results indicate that the improved algorithm produces smaller rating errors with larger data volumes. This means that more rating data may yield more accurate results, though overfitting should be prevented.

**c) Real-time performance under different incremental data scales.** As shown in [Figure 9: see original paper], the horizontal axis represents the scale of real-time data stream input, and the vertical axis represents time, reflecting the impact of data scale on incremental model computation time.

The data in the figure represents the number of rating entries flowing into the recommendation system per second. For a cluster of five PCs configured with Core i5 quad-core 3.5GHz processors and 8GB memory, the results are quite satisfactory. Relying on powerful processing capabilities and real-time computation mechanisms, the scalability issues of collaborative algorithms are significantly improved. In practice, real-time performance will further improve as cluster scale expands.

**d) Precision and recall under different data scales.** With fixed algorithm model parameters, [Figure 10: see original paper] shows the variation curves of recommendation result precision and recall across different data scales.

In [Figure 10: see original paper], triangles represent precision, squares represent recall, and circles represent the harmonic mean of precision and recall. The results show that recommendation precision increases with growing rating data volume, while recall continuously decreases, and F1 slowly increases. This indicates that under constant model parameters, precision and recall exhibit a

negative correlation trend to some extent, consistent with previous speculation: accuracy improves with more rating data, but novelty decreases accordingly. However, F1 continues to increase with data volume.

---

## 4 Conclusion

This algorithm embodies ensemble learning ideas from both algorithmic and platform perspectives, which can be summarized as follows:

**a) At the algorithm design level.** To reduce the time complexity of similarity computation, decrease rating sparsity, and increase the possibility of distributed parallel computing, the algorithm employs item popularity and rating quality factors to adjust and optimize user similarity and item similarity, making them closer to actual similarity. For rating prediction, the algorithm uses TopN neighbor similarities and ratings weighted summation to calculate each user' s predicted rating list. The two recommendation lists are then weighted and summed with the least squares recommendation list to obtain the final recommendation list. The three weights represent different dimensional weights that can be dynamically adjusted. If certain dimensional characteristics need emphasis, real-time parameter adjustment can achieve optimal recommendation parameters. If discovering certain user preferences is desired, these dimensional weights can also be adjusted.

**b) At the algorithm execution platform level.** Based on the Spark distributed computing platform and utilizing Spark Streaming' s stream processing mechanism, an incremental recommendation model combining multiple collaborative filtering methods is implemented, including similarity incremental models and recommendation incremental models. The similarity incremental model continuously merges new similarities with old ones for persistence, where incremental computation does not affect persisted quantities. The recommendation model follows the same principle, except that recommendation content updates depend on similarity updates for each user.

The experimental data uses public MovieLens and Netflix movie rating datasets. Horizontally, the algorithm optimization and design are relatively reasonable; vertically, the algorithm performs better than expected on the given test sets. Experimental results demonstrate that compared with previous similar recommendation algorithms, the improved algorithm achieves better personalization, smaller prediction errors, more real-time recommendations, and more flexible scalability, showing varying degrees of improvement in rating prediction, accuracy, and scalability.

This hybrid collaborative filtering algorithm based on algorithm integration is particularly suitable for recommendation scenarios based on user profiling, such as product recommendations, music recommendations, food recommendations, travel recommendations, and many other domains.

## References

[1] Francesco Ricci, Lior Rokach, Bracha Shapira, et al. Recommender systems handbook [M]. New York: Springer, 2011, 1 (1): 39-184
[2] Cheung K W, Tian L F. Learning user similarity and ratings for collaborative recommendation [J]. Information Retrieval, 2004, 7 (3-4): 395-410
[3] Balabanovic, M. Shoham. Content-based Collaborative Recommendation [J]. Communications of the Association for Computing Machinery, 1997, 40 (3): 66-72
[4] Wang Cheng, Zhu Zhigang, Zhang Yuxia, et al. Recommendation efficiency and personalization improvement of user-based collaborative filtering algorithm [J]. Small Microcomputer Systems, 2016, 37 (3): 428-432
[5] Tan Yunzhi, Zhang Min, Liu Yiqun, et al. Collaborative recommendation framework based on user ratings and review information [J]. Pattern Recognition and Artificial Intelligence, 2016, 29 (4): 359-366
[6] Zhang Yu, Cheng Jiujun. Study on recommendation algorithm with matrix factorization method based on MapReduce [J]. Computer Science, 2013, 1 (1): 19-23
[7] Koren Y. Factorization meets the neighborhood: a multifaceted collaborative filtering model [J]. Procedia Computer Science ACM, 2008: 426-434
[8] Deshpande M, Karypis G. Item-based top-N recommendation algorithms [J]. ACM Trans on Information Systems, 2004, 22 (1): 143-177
[9] Linden G, Smith B, York. Amazon recommendations: Item-to-item collaborative filtering [J]. IEEE Internet Computing, 2003, 7 (1): 76-80
[10] Wu Yitao, Zhang Xingming, Wang Xingmao, et al. Collaborative filtering algorithm based on user fuzzy similarity [J]. Journal of Communications, 2016, 37 (1): 198-206
[11] Xin Xinrui, Meng Caixia, Zhou Wen, et al. An improved singular value decomposition recommendation algorithm based on local structure [J]. Journal of Electronics and Information Technology, 2013, 35 (6): 1284-1288
[12] Hu Jun, Hu Xiande, Cheng Jiaxing. Big data hybrid computing model based on Spark [J]. Computer System Applications, 2015, 24 (4): 143-150
[13] Principles and applications of Kafka message management system [DB/OL]. http://kafka.apache.org/documentation/#introduction
[14] Spark streaming mechanism and overview [DB/OL]. http://spark.apache.org/docs/latest/streaming-programming-guide.html
[15] HBase columnar distributed database storage principles and applications [DB/OL]. https://hbase.apache.org/book.html
[16] Chen Jirong, Le Jiajin. Overview of big data solutions based on Hadoop ecosystem [J]. Computer Engineering and Science, 2015, 37 (10): 35-42
[17] Koren Y. Factorization meets the neighborhood: a multifaceted collaborative filtering model [J]. ACM Transactions on Knowledge Discovery from Data, 2008, 2 (1): 1-34