Time-Based Distributed Collaborative Filtering Recommendation Algorithm

Qiao Li[1,2,3], Xiantong Hu[1], Linfei Zhou[1], Xiao Zheng[1,2,3], Wei Zhao[1,3(✉)], Yunquan Gao[1,3], and Xuangou Wu[1,3]

[1] School of Computer Science and Technology, AnHui University of Technology, Maanshan 243032, China
{qiaoli,xzheng,wuxgou}@ahut.edu.cn, zlf0001@mail.ustc.edu.cn, gaoyunquan@bupt.edu.cn
[2] Hefei Comprehensive National Science Center, Institute of Artificial Intelligence, Hefei 230026, China
[3] Anhui Engineering Laboratory for Intelligent Applications and Security of Industrial Internet, AnHui University of Technology, Maanshan 243032, China

Abstract. Recommendation systems based on collaborative filtering are widely used in many fields. Alternating Least Squares (ALS) in the Mlib Library is a distributed and parallel algorithm in Spark framework, which can solve the problems of scalability and speedup in a limited hardware resources of stand-alone systems. However, it does not consider the influence of the factor of time on the recommendation accuracy. Taking restaurant ratings as an example, this month ratings are more reliable than those from last year. Thus, the motivation in our proposal re-scores ratings with different time weights. We improve ALS in its process of data preparation according to requirements on the structure of data input. Consequently, our improvement does not need to modify the main body of ALS. Experimental results validate effectiveness that our proposal outperforms the original ALS in recommendation accuracy.

Keywords: Collaborative filtering · ALS · Time factor · RMSE

1 Introduction

With the development of the Internet, the problem of information overload has become increasingly serious. The emergence of the recommendation system can help people to obtain the interested information [1]. The collaborative filtering algorithm has been widely applied in various recommendation systems.

In recent years, parallel and distributed computing models have been used in recommendation systems [2], which not only speeds up the recommendation process, but also has a good scalability. For example, many researches exploit Hadoop and Spark to improve the parallelism of recommendation system operations [3–6]. These efforts are based on new computing models and platforms,

C. T. Calafate et al. (Eds.): MONAMI 2021, LNICST 418, pp. 245–255, 2022.
https://doi.org/10.1007/978-3-030-94763-7_19

which parallels many classic machine learning algorithms such as apriori, k-means, and so forth. However, the existing research mainly studied the process of matrix decomposition and the iterative phase to improve the algorithm of Alternating Least Square (ALS), from perspective of feature vectors in post-evaluation results. However, the consideration of parallel execution of data pre-processing in the early stage is not enough. Algorithm accuracy, as well as efficiency, can be improved by means of pre-processing data, such as normalization. Thus, this work has the following considerations: We improve the original ALS algorithm within the Spark platform by exploiting influence of the time factor to adjust dataset scores. This process is in the data preparation according to requirements on the structure of data input. Experimental results validate effectiveness of our proposal in terms of the accuracy of Root Mean Square Error (RMSE).

2 Related Work

By exploiting the PySpark framework and Resilient Distributed Datasets (RDD) in Spark, Bhowmick improved existing association rules Apriori algorithm [5]. The datasets with different structures are used to obtain the data analysis experiment results, and the improved method is verified to get a better level of performance. In [6], a parallel computing ALS acceleration algorithm (ALS-NCG) is proposed, in which the performance of running and convergence times is improved. In [7,8], the authors compared advantages and disadvantages in various performance indicators (RMSE, MAE, etc.) during the execution of the collaborative filtering algorithm implemented in Hadoop platform and the Spark platform. It discuss the impact of multiple parameter on recommendation results. And it also emphasized the advantages of memory-based computing frameworks. In [9], users and items are related by the concept of incorporating labels. Then, through the continuous increase of processing nodes, the cold start problem of collaborative filtering is alleviated, and cold start, scalability, throughput and data sparseness in different scenarios are discussed. The literature [10] designed a new loss function by calculating the similarity between users and items, thus improving the accuracy of the evaluation index RMSE. The literatures [9,10] mainly focused on the inherent correlation between users and items, which improves the performance of the entire execution process. Literature [11] propose a hybrid collaborative filtering distributed execution model specifically, it uses the recommended results as input to KMeans clustering algorithm, and verifies scalability, execution time and robustness to demonstrate its effectiveness. Literatures [10,11] considered the optimization of one-step process of parallel execution. Its main point to improve the response time of the recommendation system, and the proposed new model and algorithm also had effects on improving cold start and scalability. In the recommendation system, a very important step is matrix decomposition. A basic version of matrix decomposition has been completed in the Spark framework. Literature [12] designed multiple regular term formulas to derive five different models, considering the prediction of data values in a specific scenario. It analysed missing data under the Spark platform and verified the rationality and effectiveness of the proposed model through

analysis of execution time of different sizes of data set and RMSE. Literature [13] describes a new collaborative filtering recommendation algorithm based on probability matrix factorization. Finally, time windiows weighting is integrated into the rating matrix to construct the 3D user-item-time model. The proposed algorithm achieves better overall performance than other algorithms.However, the experimental results in spark parallel framework are not considered in this paper. The above studies [3,4,7,8] mainly focused on the evaluation of performance improvement by using the new framework platform. Both the literature [6] and this paper study ALS algorithms, but we underscore the acceleration and optimization of matrix decomposition. The literature [9–12] mainly focused on step optimization in the matrix decomposition stage, and [10,11] accelerated the system execution by proposing new models and algorithms, such as considering the correlation between users and items. The existing researches mainly focused on the optimization of the matrix decomposition phase of the ALS algorithm. This paper starts from the perspective of the initial stage of data preprocessing. It improves the performance by designing a distributed ALS algorithm based on time factor.

3 Problem Description

3.1 Matrix Factorization

The most common matrix factorization technique is evolved from SVD. Suppose there are m users and n items in total, then the user-item-rating matrix is $P \in \mathbb{P}^{(m \times n)}$, where the rating represents the user's preference for items. This matrix is usually very sparse, most of the scoring elements are missing, and our task is to predict the missing values in it. For each user u set a user vector $x_u \in \mathbb{P}^f$, and each item i set an item vector $y_i \in \mathbb{P}^f$, then the predicted value is expressed as the inner product of the two $\widehat{r}_{ui} = x_u^T y_i$. Then the objective function is:

$$L = \min_{x_*, y_*} \sum_{r_{u,i}} (r_{ui} - x_u^T y_i)^2 + \lambda (\sum_u || x_u ||^2 + \sum_i || y_i ||^2) \tag{1}$$

SGD is an iterative method (iterative method), while the other method, the protagonist of this article ALS, is a direct method. Since both x_u and y_i in Eq. (1) are unknown, the function is non-convex and it is difficult to optimize directly. However, if all yis are fixed and regarded as constants, then Eq. (1) becomes a least squares problem about x_u, and the analytical solution can be obtained directly. So you can fix y_i to find x_u, and then fix x_u to find y_i. The two alternate continuously, and this process repeats until it converges. Therefore, ALS is called alternate least squares (alternating least squares), which is actually a bit similar to the alternate solution of E step and M step in the EM algorithm.

The following is a detailed derivation of the ALS algorithm flow. The user vector $x_u \in \mathbb{P}^f$ and the item vector $y_i \in \mathbb{P}^f$ have been defined above, then all users can be combined into a matrix $X \in \mathbb{P}^{(m \times f)}$, and all items are combined

into a matrix $Y \in \mathbb{P}^{(n\times f)}$, The entire scoring matrix is $P = XY^T \in \mathbb{P}^{(m\times n)}$. Then find the partial derivative of Eq. (1) fixed Y with respect to x_u:

$$\frac{\partial L}{\partial x_u} = -2 \sum_{i \in r_{u*}} (r_{ui} - x_u^T y_i) y_i + 2\lambda x_u = 0 \tag{2}$$

$$\begin{aligned} x_u &= (\sum_{i \in r_{u*}} y_i y_i^T + \lambda I)^{-1} \sum_{i \in r_{u*}} r_{ui} y_i \\ &= (Y_u^T Y_u + \lambda I)^{-1} Y_u^T P_u \end{aligned} \tag{3}$$

Among them, r_{u*} represents all items rated by user u, $Y_u \in \mathbb{P}^{|r_{u*}|\times f}$ represents a matrix of all items rated by user u, and $P_u \in \mathbb{P}^{|r_{u*}|}$ represents a vector of all items rated by user u. In the same way, fix X to obtain the partial derivative of y_i:

$$\begin{aligned} y_i &= (\sum_{u \in r_{*i}} x_u x_u^T + \lambda I)^{-1} \sum_{u \in r_{*i}} r_{ui} x_u \\ &= (X_i^T X_i + \lambda I)^{-1} X_i^T P_i \end{aligned} \tag{4}$$

According to Eq. (1), the partial derivatives of x_u and y_i are obtained, and Eq. (3) and Eq. (4) are obtained. It can be noted that the calculation of each x_u and y_i of Eq. (3) and Eq. (4) are independent, witch can be calculated in parallel to increase the speed.

3.2 System Model

User's scoring behavior on an item in recommendation systems can be expressed as a matrix $P_{(m\times n)}$, indicating scores of n items by m users. $P_{(m\times n)}$ is a sparse matrix with "missing values". One reason in that the labelling process is complex and costly. Model-based collaborative filtering algorithms predict the "missing value" from existing ratings, in a matrix completion process. The ALS algorithm is used to complete $P_{(m\times n)}$ by dimension reduction in training models. $P_{(m\times n)}$ is composed of two low-dimensional matrices $U_{(m\times k)}$ and $V_{(n\times k)}$. The minimum squared error loss function $L(U, V)$ is used to compute U and V while avoiding the overfitting problem.

The original ALS algorithm aims to decompose $P_{(m\times n)}$. Traditional matrix decompositions are difficult to handle huge data given a large size of $P_{(m\times n)}$. It computes $U_{(m\times k)}$ and $V_{(n\times k)}$ in Eq. (5),

$$P_{(m\times n)} \approx U_{(m\times k)} V_{(n\times k)}^T \tag{5}$$

where k is the minimum of m and n, which refers to the correlation dimension. It is also known as "implicit semantic factor".

The objective is to minimize the squared error loss function in Eq. (6),

$$L(U, V) = \sum (r_{u,i} - x_u^T y_i)^2 \tag{6}$$

where $r_{u,i}$ represents the rating of item i by user u, and $x_u^T y_i$ the approximate rating of item i by user u. A regularization term is applied to avoid overfitting in Eq. (7),

$$L(U,V) = \sum (r_{u,i} - x_u^T y_i)^2 + \lambda(\mid x_u \mid^2 + \mid y_i \mid^2) \tag{7}$$

where λ is a regularization coefficient. The specific solution for ALS is similar with the gradient descent method. The calculation process is as follows.

1. Set initial values V and U
2. Find the partial derivative of function $L(U,V)$ for x_u, and let the partial derivative equal to 0 to get:

$$x_u = (V^T V + \lambda I)^{-1} V^T P_u \tag{8}$$

3. Similarly, it is expected to get Eq. (9), where P_u is the p-th row, P_i the i^{th} column of P, and I $k \times k$-dimensional matrix.

$$y_i = (U^T U + \lambda I)^{-1} U^T P_i \tag{9}$$

4. Continue steps 2 and 3 until convergence.

3.3 Time Factor

Our work considers time factor in the training model. In addition, we consider the fact that the latest users have a bigger weight in the evaluation. A weight coefficient in Eq. (10) adjusts user ratings from viewpoint of time drift.

$$\theta = \frac{t_{u,i} - t_{min}^i}{t_{max}^i - t_{min}^i} \tag{10}$$

where $t_{u,i}$ refers to the current time for user u to rate item i, t_{max}^i the last time when all users rate item i, and t_{max}^i the earliest time. The weight increases with the rating timestamp, implying the rating importance. Then we rearrange the rating in Eq. (10) by Eq. (11),

$$R_{u,i} = r_{u,i} \times (1 + \theta) \tag{11}$$

where $r_{u,i}$ the original user rating. Different periods of user ratings are applied in traditional methods. Notice that it ignores influence of weights of different periods on ratings that fluctuate over time. It happens frequently that movies with low ratings at initial releasing periods become classicical along with time. Thus, the time factor should be one of criteria on ratings.

4 Algorithm Implementation

4.1 Algorithm Design

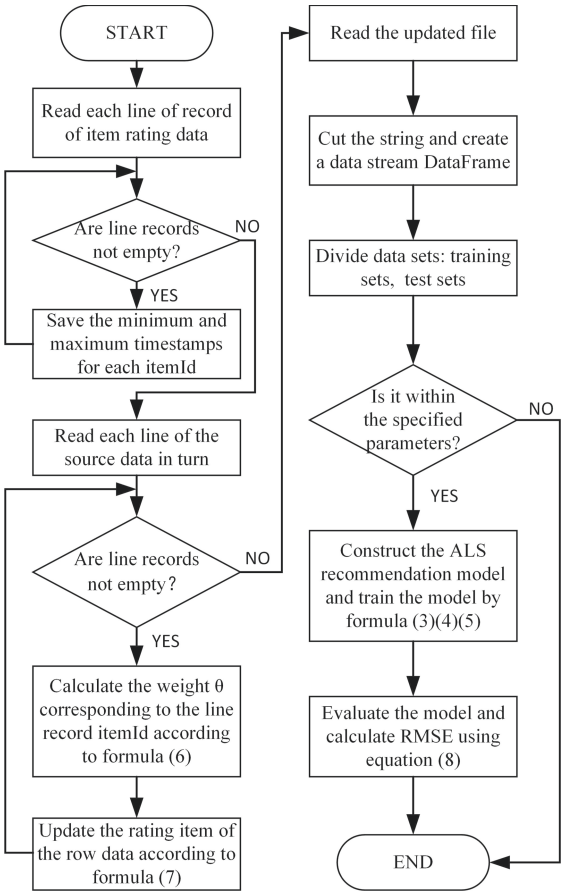

Fig. 1. Algorithm implementation flow chart

Figure 1 shows the process of our algorithm. It includes four components: to preprocess data, to find the minimum and maximum timestamps of all user scores corresponding to each item, to traverse each line of records, and to update scores in the original data by means of Eq. (10) and Eq. (11). These operations can be addressed in the Spark framework. The model is optimized multi-dimensionally with different parameters, and finally the prediction and model evaluation are performed. The specific process and implementation will be described in Algorithm 1.

4.2 Data Processing

The score dataset is processed according to Eq. (10) and Eq. (11). The timestamp in the original dataset is associated with the user's rating. Algorithm 1 shows the data processing.

Algorithm 1: Data Processing

Input: Dataset D
Output: Dataset S considering time weight
1: D = {UserID, ItemId, Rating, TimeStamp}
2: for each line in D:
3: i ← itemId
4: if TimeStamp < min_TimeStamp(i):
5: min_TimeStamp (i) ← TimeStamp;
6: if TimeStamp > max_TimeStamp (i):
7: max_TimeStamp (i) ← TimeStamp;
8: end for
9: Obtain max_TimeStamp and mix_TimeStamp according to itemId.
10: for each line in D:
11: Obtain the TimeStamp field in the raw data
12: set $t_(u,i)$ =TimeStamp.
13: Set the time weight: $\theta = \frac{t_{u,i} - t^i_{max}}{t^i_{max} - t^i_{min}}$.
14: Update the original TimeStamp according to the time weight.
15: Updating TimeStamp: $R_{u,i} = r_{u,i} \times (1 + \theta)$.
16: end for
17: Save $R_{u,i}$.

The input is dataset D to be processed, and the output is the dataset S after the correction of score items according to time weight. Lines from 5–8 obtain the minimum timestamp and the maximum timestamp of each itemId score through circulating each row in dataset D. Lines from 13–16 are to calculate the time weight according to Eq. (10). Line 17 updates scores based on the time weight.

4.3 Recommendation Algorithm

We design a recommendation algorithm to train a model with the goal of the minimum prediction error. We improve the traditional ALS algorithm in Spark in its process of data preparation according to requirements on the structure of data input. Evaluation is performed on the accuracy of RMSE.

In Algorithm 2, the parameter maxIter is the number of iterations at runtime. Each iteration can improve the accuracy of the algorithm. Rank is the number of implicit semantic factors in the model, that is, the number of features. RegParam is a regularization parameter of ALS, indicating the magnitude of λ in Eq. (7).

The algorithm input is the dataset S from Algorithm 1. The output is the prediction error. Lines 1–6 create the data reading specification and the dataset is divided into a training set and a test set. Lines 7 to 11 optimize the ALS

model by iterating. Lines 12 to 15 test the training model with the test set and evaluate the model to obtain the error.

Algorithm 2: Recommendation Algorithm

Input: Dataset S obtained in Algorithm 1
Output: Prediction error
1: Get the sc object:spark = SparkSession.builder.
2: Read file = S [userId, itemId, newRating, timestamp]
3: Create a Rating type [Int, Int, Float, Long] and convert each line to Rating.
4: Divide fields according to the Separator
5: Create DataFrame by ratings RDD.
6: Split the DataFrame into training and test.
7: for rank or lambda ∈ [m,n]
9: for maxIter ∈ [m,n]
10: Set ALS parameters(rank=rank, maxIter=maxIter, regParam=lambda, userCol="userId", itemCol="movieId", ratingCol="rating", coldStartStrategy="drop").
11: Train model by training data.
12: Predictive model.
13: Evaluate model and Get the error.
14: Write the error to text.
15: end for
16: end for
17: Stop Spark.

5 Performance Evaluation

5.1 Datasets and Evaluation Metrics

This experiment is to predict user ratings for each movie. In actual application scenarios, the movie with the highest predicted ratings will be recommended to the user. The experiment uses dataset Movielens with 100,000 and 1 million film records. Each row of dataset records contains four attributes: User ID, Movie ID, User Rating, and Timestamp. We apply the metirc RMSE in Eq. (12) to evaluate our model. It is arithmetic square root of mean square error to measure how much the data changes. Note that prediction accuracy increases inversely with RMSE.

$$RMSE = \sqrt{\frac{\sum_{u,i}(r_{u,i} - p_{u,i})^2}{\mathsf{N}}} \tag{12}$$

where N represents the number of experiment samples, $p_{u,i}$ and $r_{u,i}$ the predicted and observed scores of user u for movie i, respectively.

5.2 Experiment Settings

The purpose of this experiment is to verify the advantages and disadvantages of an improved algorithm considering time factor. The experiment is performed

in the PySpark cloud environment which consists of five Alibaba Cloud Server ECSs. One is a master node and the other four are slave nodes.

In collaborative filtering algorithms, the key is to train model parameters accurately. The algorithm has the following parameters: Rank, maxIter, and λ. Rank, the number of implicit semantic factors, has an influence on the fitness of the training model in the test model. MaxIter is the number of iterations. λ is the regularization parameter of ALS. Three parameters are independent variables to observe RMSE.

5.3 Evaluation

Figures 2(a) and (b) are three-dimensional maps of RMSE. X-axis in the figure is the number of the implicit semantic factor Rank from 1 to 10. Y-axis is the number of iterations maxIter from 1 to 10. The Z-axis is RMSE.

The results demonstrate that the improved algorithm has a significant improvement in RMSE. It can be seen that RMSE decreases sharply with Rank. It indicates that the increase of the Rank value results in improvement of performance. RMSE in two algorithms first decreases rapidly, and then becomes stable. Specifically, RMSE fluctuates slightly while Rank changes from 3 to 10, which indicates that Rank should not be too large. In addition, RMSE in Fig. 2(a) is much larger than RMSE in Fig. 2(b) when maxIter is from 1 to 3. This shows that the model is more stable given millions of data.

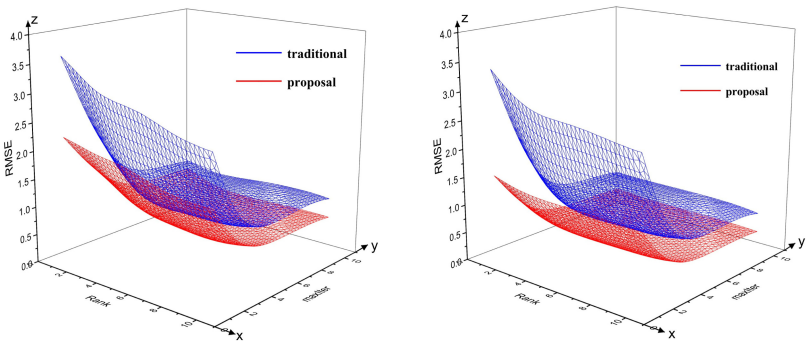

Fig. 2. (a) RMSE in 3D map. (b) RMSE given a million-level datase

With 100,000 and a million levels of data, RMSE in the proposed algorithm is smaller than that in the traditional algorithm. It is observed that RMSE tends to converge to a Rank of 10. Therefore, we fix Rank = 10 to observe RMSE by alternating λ and maxIter in Figs. 3(a) and (b).

The X-axis in Figs. 3(a) and (b) are λ in [0.0001, 0.0003, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3]. The Y-axis is the number of iterations maxIter from 1 to 10,

and the Z-axis is RMSE. We have the same result that the proposed algorithm is better than the traditional algorithm. In Fig. 3(a), RMSE gradually decreases with λ from 0 to 0.5. When the λ is close to 0.5, the RMSE reaches its lowest value. Afterwards, RMSE increases with λ. In summary, the proposed algorithm considering time factor is more effective than the traditional algorithm from the respect of RMSE.

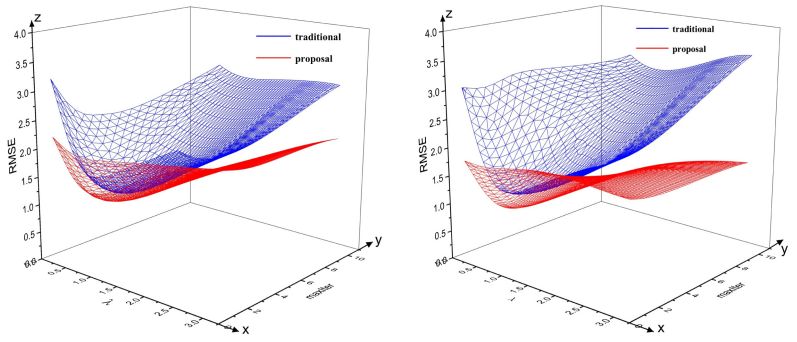

Fig. 3. (a) RMSE given a 100000-level datase. (b) RMSE given a million-level dataset

6 Conclusion

This paper proposes a time-based distributed parallel recommendation algorithm in the PySpark platform. The results show that our proposal outperforms the collaborative filtering algorithm and it reaches stable fast. We have verified its effectiveness on RMSE with multiple parameters including the number of implicit semantic factors, the number of iterations, and the regularization term.

Acknowledgement. This work was supported in part by the National Natural Science Foundation of China under Grants 61672038 and 61702006, in part by the Major Technologies R&D Special Program of Anhui under Grant 16030901060. And in part, the Program for Synergy Innovation in the Anhui Higher Education Institutions of China (Grant No. GXXT-2020-012).

References

1. Yang, X., Guo, Y., Liu, Y., Steck, H.: A survey of collaborative filtering based social recommender systems. Comput. Commun. **41**, 1–10 (2014). https://doi.org/10.1016/j.comcom.2013.06.009
2. Karydi, E., Margaritis, K.: Parallel and distributed collaborative filtering: a survey. ACM Comput. Surv. (CSUR) **49**(2), 37 (2016)

3. Gousios, G.: Big data software analytics with apache spark. In: Proceedings of the 40th International Conference on Software Engineering: Companion, pp. 542–543. Association for Computing Machinery, New York (2018)
4. Meng, X., et al.: MLlib: machine learning in apache spark. J. Mach. Learn. Res. **17**(1), 1235–1241 (2016)
5. Gao, F., Bhowmick, C., Liu, J.: Performance analysis using apriori algorithm along with spark and python. In: Proceedings of the 2018 International Conference on Computing and Big Data, pp. 28–31. ACM (2018)
6. Winlaw, M., Hynes, M.B., Caterini, A., De Sterck, H.: Algorithmic acceleration of parallel ALS for collaborative filtering: speeding up distributed big data recommendation in spark. In: 2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS), pp. 682–691. IEEE (2015)
7. Kupisz, B., Unold, O.: Collaborative filtering recommendation algorithm based on hadoop and spark. In: 2015 IEEE International Conference on Industrial Technology (ICIT), pp. 1510–1514. IEEE (2015)
8. Verma, A., Kumar, D.: Evaluating and enhancing efficiency of recommendation system using big data analytics (2017)
9. Panigrahi, S., Lenka, R.K., Stitipragyan, A.: A hybrid distributed collaborative filtering recommender engine using apache spark. Procedia Comput. Sci. **83**, 1000–1006 (2016)
10. Xie, L., Zhou, W., Li, Y.: Application of improved recommendation system based on spark platform in big data analysis. Cybern. Inf. Technol. **16**(6), 245–255 (2016)
11. Lenka, R.K., Barik, R.K., Panigrahi, S., Panda, S.S.: An improved hybrid distributed collaborative filtering model for recommender engine using apache spark. Int. J. Intell. Syst. Appl. **10**(7), 74 (2018)
12. Harper, F.M., Konstan, J.A.: The movielens datasets: history and context. ACM Trans. Interact. Intell. Syst. **5**(4), 19 (2016)
13. Zhang, P., Zhang, Z., Tian, T., Wang, Y.: Collaborative filtering recommendation algorithm integrating time windows and rating predictions. Appl. Intell. **49**(8), 3146–3157 (2019). https://doi.org/10.1007/s10489-019-01443-2