Collaborative Filtering Recommendation System Implemented in Apache Spark

Nan Li

nan.li@uwaterloo.ca

Abstract. Recommendation system has been widely used in E-commerce to provide customized service to individuals for a long time. In this paper, we try to find out how long it will take to train a recommendation model on a huge real-world dataset with a distribution computing engine; what size of dataset is suitable for a high-quality recommendation model; how well the performance of popular recommendation models is. In our experiments, we apply two recommendation models of collaborative filtering with Spark machine-learning libraries on Yelp open dataset through cross validations. Then we compare the recommendation performance and training time of the two algorithms on different sub datasets. Based on the experiment results, we draw several conclusions to answer the proposed questions.

1 Introduction

In the era of information explosion, people always feel overwhelmed by the amount of data when visiting the Internet, especially the e-commerce websites, such as Netflix or Amozon. These e-commerce retailers make a large number of deals including music, movies, electronic goods with billions of consumers everyday. In this kind of on-line business, recommender system plays a key role in providing people with personalized services, which can help customers find items of their own interests, and enhance their satisfaction and loyalty to the retailers. Generally speaking, there are two main recommendation strategies: content-based filtering and collaborative filtering(CF) [11]. They are both widely used in commercial recommendation systems. Besides recommending methods, training a recommending model on a large-scale commercial dataset within a feasible time is considered extremely important. Most on-line retailers applied parallelizing the training with distributing computing engine to speed the process. Apache Spark framework is one of the extensively used cluster computing engine, integrating multiple functions including machine learning.

1.1 Recommendation Algorithms

As one of the main recommendation algorithms, Content-based filtering algorithm creates users or products' profiles of intrinsic characteristics and recommends based on items' properties and users' preference. For example, a music

profile may include features about its genre, artists or rank of popularity. Then, recommendations can be made to match the music's attributes to consumers' preference. One disadvantage of this system is that it needs to learn the specific attributes of products or users. Sometimes it is not easy to gather enough information to make comprehensive profiles [11, 17].

Unlike content-based filtering, the major appeal of collaborative filtering is its domain-free attribute, making use of the interactions between consumers and products, such as users' dislikes and likes on some products as shown in Figure 1. It assumes similar consumers will have similar rating for the same items, and people with similar tastes in the past will also be fond of the same in the future. This method relies on a large amount of information on users' past behavior, including previous transactions or product ratings, and make predictions based on relationships between users and products. CF algorithms have been proved to be more accurate than content-based filtering methods if sufficient preference data available [8, 11, 16].

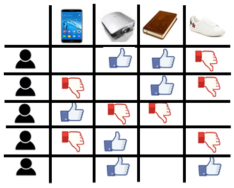


Fig. 1. The interactions between consumers and products.

User preference data can be categorized into two types: explicit feedbacks and implicit feedback. Explicit feedbacks are typically in forms of users' ratings on items. However, for plenty of businesses there are no such explicit ratings, but other types of feedbacks, such as users' views, purchases, likes, clicks. These kinds of feedbacks are considered as implicit feedbacks, which indirectly indicate people's preferences. Obviously, implicit feedbacks are less accurate than explicit ones. Some algorithms are proposed to convert the implicit feedbacks into numbers to represent users' confidence levels of consumption [8, 9].

Although collaborative filtering generally gives rise to accurate recommendations, it suffers from a so-called "cold start" problem, which refers to new users or items without any transaction history. In this aspect, content-based filtering is usually adopted [7, 11] to solve it.

1.2 Apache Spark Framework

Apache Spark computing system is a distributing computing engine, usually running in clusters to process large-scale data efficiently. It provides high-level APIs in different languages including Scala, Java, Python and R, and integrates four main modules: Spark SQL, MLib, Spark Streaming and GraphX as shown

in Figure 2. Its machine learning libraries provide a bunch of common-used algorithms and methods [3, 4].

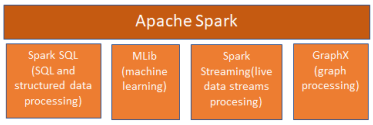


Fig. 2. Spark Main Modules.

The main data type Spark provides is resilient distributed dataset (RDD), which can be partitioned by Spark Engine across the nodes of clusters and operated in parallel with fault-tolerant ability. There are two ways to create RDDs: using "parallelize" function to make a raw data type (Array or List) become RDD, or reading data from a storage system including traditional file systems, HDFS, HBase. One characteristic of RDD is its lazy evaluation, which means the results of RDDs are not computed right away when operating transformations, such as a "join" method. Instead, Spark engine will record all the transformations applied to the RDD. When an action is required to get the result of the RDD, for example "collect", all the previous transformations will be computed together. In this way, Spark can process more efficiently by optimizing the transformations altogether [5]. In a recent version of Spark, a new interface DataFrame was added to provide high-level database-like operations.

Currently, there are three types of cluster managers for Spark: Spark's own standalone cluster manager, Apache Mesos and Hadoop YARN. Mesos and YARN need to integrate other systems to Spark, which are always implemented in commercial applications [1].

1.3 Organization

In this paper, we focused on comparing recommendation qualities of two collaborative filtering methods implemented by Spark libraries on different subsets on a large real-world dataset with explicit rates, and try to find the relation between the size of dataset and the accuracy of the prediction. Also, we observed the running time of the models being trained on a cluster consisting of two Spark workers. The rest of the paper is organized as follows. Section 2 describes the CF algorithms used in this paper. In following section 3 and section 4, we present how our experiments were executed and their results. The final section provides some conclusions of our work.

2 Collaborative Filtering Models

CF algorithms can be divided into two primary areas: neighborhood methods and latent-factor methods. Neighborhood approaches compute the similarity

(e.g., cosine similarity, Pearson Correlation) between users (user-based CF) or items (item-based CF) through rating vectors, and make predictions usually as a weighted average of this similarity. An alternative approach trying to discover the relation between users and items is latent-factor models, which uncover latent features of the observed ratings. For example, in the context of movies, a latent feature may represent the movie style or genre (i.e. drama, comedy, romance). For users, each factor measures their preference on the movies [11]. Compared with neighborhood methods, latent-factor models are more scalable in processing sparse, large-scale datasets.

2.1 Related Work

There is numerous academic and industry work about these two methods since the beginning of 21st century. At first, neighborhood methods were considered as the state-of-the-art. In 2001, Sarwar et al. [14] successfully implemented an item-based CF algorithm, computing three types of similarities between items and making recommendations from both wighted sum and regression model. They also compared the performance with user-based CF, and proved that item-based CF provided better quality of predictions. Around 2007, latent-factor methods gained extensively popularity with more research work, because they showed higher accuracy in the Netflix Prize competition for the best recommendation algorithm. Huang et al. [10] compared six CF algorithms with transactional data, and evaluated the results based on five metrics in different e-commerce dataset. Zhou et al. [17] improved the ALS algorithm with weighted-λ-regulation and obtained one of the best results in the Netflix challenge for a pure method. In the meantime, Hu et al. [9] presented an improved matrix factorization algorithm to address the problem of implicit feedbacks through introducing a user confidence measurement. As the winner of Netflix Prize competition in 2007, Koren et al. [11] gave details about their method of matrix factorization which added biases variables, implicit feedbacks and temporal dynamics. After 2010, in this area, some research works tried to integrate CF with distributing computing applications (e.g. Hadoop, Apache Spark) [15, 16]; some focused on improving CF algorithms by optimizing the regression algorithm [13] or adding context-aware information[8] to improve the accuracy.

2.2 Item-based CF

Compared to user-based CF, item-based CF showed improved accuracy because the item neighborhood is more static, independent of the total number of customers or products [9, 12, 14].

Every item has a rating vector from different users, and the similarity of items are calculated based on their ratings given by the same users. The most common similarity measures in this case are: Pearson correlation, cosine vector similarity, and adjusted-cosine similarity. In this paper, we choose the cosine similarity, which is generally used. The formula of similarity between items i

and j, denoted by sim(i,j), is as follows [7, 14, 16]:

$$sim(i,j) = \frac{\sum_{k=1}^{n} R_{ik} R_{jk}}{\sqrt{\sum_{k=1}^{n} R_{ik}^2} \sqrt{\sum_{k=1}^{n} R_{jk}^2}} \tag{1}$$

where R_{ik} is the rating of user k on item i and R_{jk} is the rating of user k on item j. n indicates the number of users who co-rate both item i and item j. Then the neighborhood of item i or j can be isolated by its most similar items based on the similarity.

The prediction of the rating for an item i and a user k can be computed by the weighted average of the ratings given by user k on item i's neighbors. The prediction P_{ik} is given as [14]:

$$P_{ik} = \frac{\sum_{j=1}^{n} R_{jk} * sim(i,j)}{\sum_{j=1}^{n} sim(i,j)} \tag{2}$$

where R_{jk} is the rating of user k on the neighbor item j, and n is the number of neighbors.

2.3 Matrix Factorization Method

The most successful realizations of latent factor models are based on matrix factorization (MF) [11], which reduces the high-dimension user-item rating matrix $R_{m\times n}$ to two low-dimension latent-factor matrices $P_{m\times k}$ and $Q_{n\times k}$, as shown in Table 1,2,3. MF performs a low rank matrix approximation: $R_{m\times n} \approx P_{m\times k} Q_{n\times k}^T$, where k is the number of latent factors [13].

	item1	item2	item3	item4	item5	item6	item7
user1	4	1	?	3	5	?	2
user2	3	?	4	5	?	4	?
user3	?	2	4	?	4	5	4
user4	?	2	4	?	4	5	4

Table 1. User item rating matrix $R_{m\times n}$.

	feature1	feature2	feature3
user1	?	?	?
user2	?	?	?
user3	?	?	?
user4	?	?	?

Table 2. User latent feature matrix $P_{m\times k}$.

	feature1	feature2	feature3
item1	?	?	?
item2	?	?	?
item3	?	?	?
item4	?	?	?
item5	?	?	?
item6	?	?	?
item7	?	?	?

Table 3. Item latent feature matrix $Q_{n\times k}$.

To learn the factor matrices P and Q, the algorithm minimizes the squared error with a regularization parameter λ to avoid overfitting as shown below [11]:

$$\min_{p_m, q_n} \sum_{m,n} (r_{mn} - p_m q_n^T)^2 + \lambda(||p_m||^2 + ||q_n||^2) \tag{3}$$

There are two approaches which can solve this optimization problem: gradient descent and alternating least squares (ALS). Compared to gradient descent, ALS is better for parallelizing computation because the computation of each p_m or q_n is independent of other user or item factors [11]. Due to this quality, Spark chose ALS as its implementation of collaborative filtering.

3 Experiments

3.1 Dataset

We executed our experiments on Yelp open dataset [6], a real business dataset from customers using Yelp service. The dataset we used has 156639 businesses (items), 1183362 users and 4736896 reviews (ratings), which is on a 1 to 5 integer scale. We divided this dataset into a training set and test set, and chose 0.8 as the training/test ratio value which was commonly used in previous literatures and proved to be the optimum value [7, 14].

3.2 Apache Spark Environment Setup

The experiment environment is a cluster consisting of two virtual machines from Google Cloud with 8 cores(2.20GHz) and 30G memory respectively. The Apache Spark framework(the latest version 2.3.0) was deployed on this cluster in standalone mode with one master and two slave workers. We can check the deployment and the job running situations from its web-UI portal as shown in Figure3.

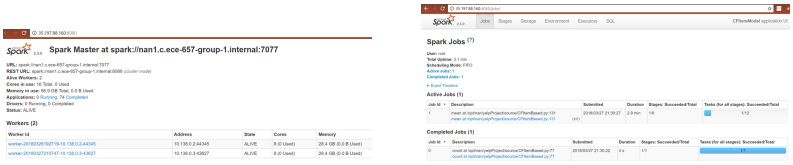


Fig. 3. The plot of predict elements with age and job dimensions.

3.3 Evaluation Metrics

We chose Root Mean Square Error(RMSE) as the evaluation metric, which was also used in the Netflix prize, to describe the deviation of predicted ratings from

their true values. The equation is as follows:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(p_i - q_i)^2}{n}} \tag{4}$$

where p_i is the true value and q_i is the predicted value. n is the number of samples.

3.4 Preprocessing

We implemented some data preprocessing on the Yelp dataset to make it suitable for our experiments. First, the dataset was converted from Json format to CSV format, which is much easier for Spark MLlib to process. The original string-format userID and businessID were mapped to integers, because Spark CF methods only support integers for user and item ids. Then, we chose 10 cities with the most businesses from the whole dataset to test the algorithms. Through this, we can evaluate the recommendation accuracy on datasets of different sizes. Also, since every city has its own characteristics or "gene", such as the local cuisine, it makes sense to build and customize recommendation models on every city dataset. We used Spark Dataframe APIs to join the user set, business set and review set together to a full rate set. In this way, we also made sure that all the processed review data has valid userID and businessID. At last, we applied filter function to get 10 cities sets from the full rate set. The city names and rating numbers are shown below and in Figure 4.

```
1492738 | Las Vegas     162442 | Pittsburgh   4736896 | All
 519312 | Phoenix       114613 | Montral
 391848 | Toronto       117981 | Mesa
 212373 | Charlotte     147718 | Henderson
 279929 | Scottsdale    149078 | Tempe
```

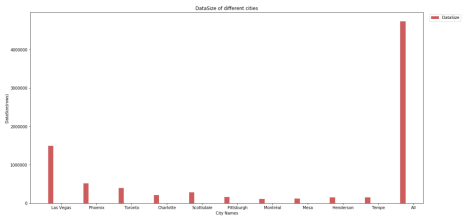


Fig. 4. The plot of data size of 10 cities datasets and the whole dataset.

3.5 ALS Experiment

The Spark MLlib has provided an ALS model for the matrix factorization of collaborative filtering [2], and we used this interface to train the model on 11

datasets including 10 cities and the whole set. There are two parameters for us to tune: regParam(the regularization parameter) and rank(the number of latent factors). rank is the most important parameter, which affects the complexity, training time and accuracy significantly. We implemented 3-fold cross-validation on training sets to get the best value from regParam(0.02, 0.1) and rank(20, 50, 100, 200, 300). Also, to handle the cold-start problem which may occur when splitting the training and evaluation sets, we set the "oldStartStrategy" parameter to "drop" to eliminate any rows that contain "NaN" values.

3.6 Item-Based CF Experiment

Since Spark MLib does not provide predefined functions for Item-Based CF, we implemented this algorithm ourselves using Spark RDD APIs. The specific implementation is as follows:

Algorithm 1 Item-Based CF based on Spark RDD

Input: trainingRDD, testRDD, neighbor number
Output: rmse of test data

1: map trainingRDD x to (x.itemID, (x.userID, x.rating))
2: group x by itemID and get groupItemRdd of (x.itemID, list(x.userId, x.rating))
3: get the Cartesian product x of groupItemRdd and filter with (x: x.itemID < x.itemID)
4: map Cartesian set x to ((x.itemID1, x.itemID2), (x.item1.userRateList, x.item2.userRateList))
5: calculate cosine similarity of items and get similarityRDD((item1, item2), similarity)
6: flatten similarityRDD and get the whole similarity set with ((item2, item1), similarity) and ((item1, item2), similarity)
7: group similarityRDD by item and get the neighbors of highest similarities: itemSimilarityRdd (item1, [(neighborItem2, similarity), ...])
8: flattern itemSimilarityRdd and set neighborItem2 as key (neighborItem2, (item1, similarity))
9: join the previous RDD with trainingRDD, and get item1UserItem2InfoRdd((item1, user), (neighborItem2, similarity, rate2))
10: group item1UserItem2InfoRdd by item1 and user, and get a neighborItem list((item1, user), neighborItems)
11: compute the wighted sum of ratings and similarities from neighbors and predict rating for every user and item((user, item), predicted rating)
12: join the RDD of predicted rating with testRdd on the same user and item
13: calculate rmse of predicted rating and real rating
14: **return** rmse

In this algorithm, there is one parameter (number of neighbors) to tune, for which we executed 3-fold cross-validation to get the best value from 15, 20, 30. Since our implementation of this algorithm uses Cartesian product which is highly time and memory consuming, if applied to large dataset, the training

could take a very long time and possibly run out of the memory. To avoid this situation, we only used 5 small-size datasets to evaluate the performance.

4 Experiment Results

In this session, we present and compare our experiment results of two CF algorithms: ALS and item-based CF, including the prediction accuracy and computing performance. Based on our test results, we make some suggestions about how to make recommendation better using CF models.

4.1 ALS and Item-based CF Experiment Results

After 3-fold cross-validation experiments of training ALS model [2] on 11 datasets, the best value of the parameter regParam is unanimously 0.1. For the parameter rank, the values affect the recommendation accuracy differently on every dataset as shown in Figure 5, and the best values for every dataset are shown in Figure 6. Opposite to the assumption that the highest number of latent factors will lead to the highest test accuracy, from our experiments, it shows that every dataset, even with similar dataset size, has its own rank value, which suits this dataset best. Simply applying the rank value of one dataset to another dataset with a similar size or just choosing the highest rank value will lower the recommendation quality of a particular dataset. Every dataset needs to have its own parameters tuned to get the optimal results.

Also, we find that the largest training dataset does not assure the best predictions. The RMSE of the whole dataset with 4736896 samples is 1.498, while the accuracy of Montreal with 114613 samples is higher, with RMSE of 1.331. For CF algorithms, the local patterns of a dataset seem to be more important than a global one. It is possible that we can choose datasets from a certain group such as an area or a certain time to get a better fit model.

The training time of the models on all the datasets differs significantly as shown in Figure 5. The longest time of the whole dataset of rank 300 is 18285s, more than 5 hours. For the same dataset, with rank 200, the training time decreases to 5741, less than 1/3 of rank 300. However, the disparity of RMSE between rank 200 and 300 can almost be ignored, less than 0.0003. Due to this, we should consider the trade-offs of accuracy and computing time, and choose the number of latent factors within a feasible computation limit.

Similar to the results of ALS experiments, the best values of neighbor numbers for item-based CF vary among 5 cities as shown in Figure 7 and Figure 8. A high neighbor number may result in worse prediction for a specific set. The training time is pretty long here. Even for a small neighbor number and samples, it exceeds 5 minutes. This maybe due to the lack of optimization of our implementation.

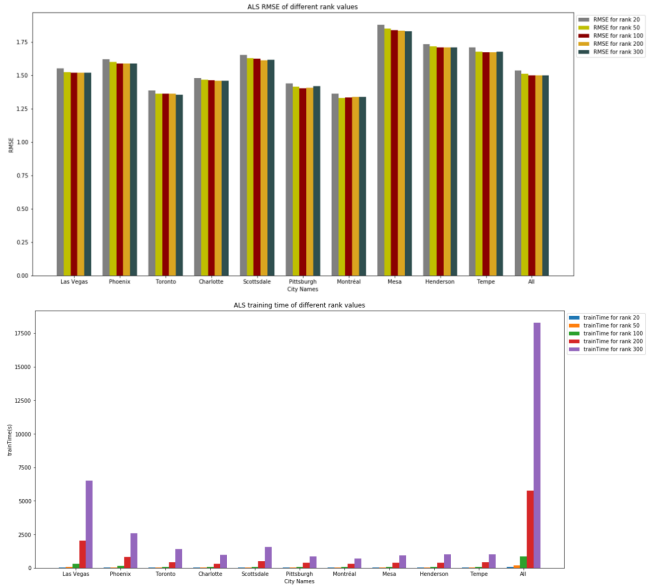


Fig. 5. ALS test results and running time on 11 datasets with different rank values.

cityName	DataSize	rank	RMSE
Las Vegas	1492738	100	1.517892
Phoenix	519312	300	1.58666
Toronto	391848	300	1.355409
Charlotte	212373	300	1.457238
Scottsdale	279929	200	1.612317
Pittsburgh	162442	100	1.402812
Montréal	114613	50	1.331522
Mesa	117981	300	1.82921
Henderson	147718	300	1.70807
Tempe	149078	200	1.672105
All	4736896	300	1.498306

Fig. 6. The best ALS test results on 11 datasets.

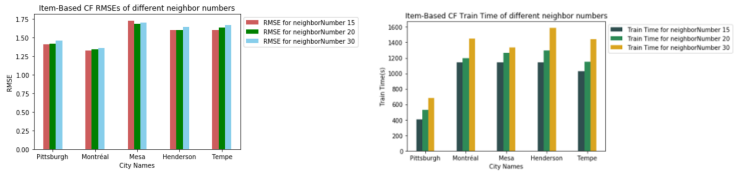
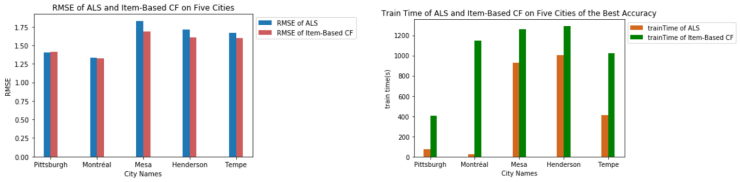


Fig. 7. Item-based CF test results and running time of 5 datasets with different neighbor numbers.

4.2 The Comparison of Two Algorithms

Comparing the recommendation of the two algorithms, for four cities, the prediction quality of Item-based CF is better than ALS; for the other one, their accuracies are very similar, as shown in Figure 8. Since it has been proved that in general, ALS model has better accuracy than neighborhood methods[8, 11, 17], if we try to get a high-quality recommendation by using Spark ALS in the product environment, optimizations are needed to Sparks predefined functions, such as the methods listed in Korens paper [11]. From our experiments, one advantage ALS model has is its training time on average is much less than item-based CF method.



cityName	DataSize	RMSE(Item-Based CF)	RMSE(ALS)	neighborNumber	rank
Pittsburgh	162442	1.41148056	1.402812273	15	100
Montréal	114613	1.322686046	1.331521863	15	50
Mesa	117981	1.68877003	1.829209913	20	300
Henderson	147718	1.603770989	1.708069733	20	300
Tempe	149078	1.598999723	1.672104933	15	200

Fig. 8. The comparison of best ALS and Item-Based CF test results and running time on 5 datasets.

There is another interesting fact. The RMSEs of Pittsburgh and Montreal for both ALS and Item-Based CF algorithms are very close. This may infer that the dataset of these two areas have some fixed patterns, perhaps because there are more similarities of local businesses there. Maybe due to this characteristics, the accuracies of these two cities are higher than other cities. It seems that we can apply different CF algorithms to a same dataset and compare the predicting performance to decide whether this dataset split has suitable characteristics for training.

4.3 Distributing Computing Performance

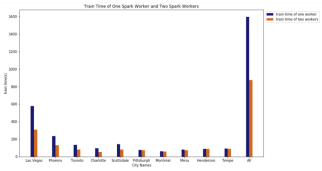


Fig. 9. The training time of one and two Spark workers for ALS model with rank 100.

We compared the training time of ALS model with rank 100 by using one Spark worker and two Spark workers. From the result shown in Figure 9, the enhancement of distributing computing is more obvious for large dataset. For the whole dataset, two workers improve the speed twice compared to one worker. In the real world, with billions of data to process, it is better to choose distribution computing frameworks to make the training more scalable.

5 Conclusion

In this paper, we experimented two widely-used CF algorithms: ALS and item-based CF through Apache Spark framework. From the observation and analysis of our tests, we came to some conclusions about how to make recommendations more effective as follows: the predefined ALS models of Apache Spark need optimizations to get better results than item-based CF; the best parameters(number of latent factors or neighbors) of both algorithms are not always the highest ones, and they should be selected based on every particular dataset even with similar size with consideration to the trade-offs of accuracy and computing time; it is possible to get a better fit model from a small local dataset with some intrinsic patterns than from a much larger general dataset; different CF algorithms can be used together to decide how to split a dataset; distributing computation is more scalable for training models.

References

1. Apache spark cluster mode overview (2018), https://spark.apache.org/docs/latest/cluster-overview.html, [last accessed 05-April-2018]
2. Apache spark collaborative filtering (2018), https://spark.apache.org/docs/latest/ml-collaborative-filtering.html, [last accessed 05-April-2018]
3. Apache spark homepage (2018), https://spark.apache.org/, [last accessed 05-April-2018]
4. Apache spark overview (2018), https://spark.apache.org/docs/latest/index.html, [last accessed 05-April-2018]
5. Apache spark rdd programming guide (2018), https://spark.apache.org/docs/latest/rdd-programming-guide.html, [last accessed 05-April-2018]
6. Yelp open dataset (2018), https://www.yelp.ca/dataset, [last accessed 05-April-2018]
7. Gong, S.: A collaborative filtering recommendation algorithm based on user clustering and item clustering. JSW 5(7), 745–752 (2010)
8. Hidasi, B., Tikk, D.: Fast als-based tensor factorization for context-aware recommendation from implicit feedback. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 67–82. Springer (2012)

9. Hu, Y., Koren, Y., Volinsky, C.: Collaborative filtering for implicit feedback datasets. In: Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on. pp. 263–272. Ieee (2008)
10. Huang, Z., Zeng, D., Chen, H.: A comparison of collaborative-filtering recommendation algorithms for e-commerce. IEEE Intelligent Systems 22(5) (2007)
11. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer 42(8) (2009)
12. Linden, G., Smith, B., York, J.: Amazon. com recommendations: Item-to-item collaborative filtering. IEEE Internet computing 7(1), 76–80 (2003)
13. Pilászy, I., Zibriczky, D., Tikk, D.: Fast als-based matrix factorization for explicit and implicit feedback datasets. In: Proceedings of the fourth ACM conference on Recommender systems. pp. 71–78. ACM (2010)
14. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item-based collaborative filtering recommendation algorithms. In: Proceedings of the 10th international conference on World Wide Web. pp. 285–295. ACM (2001)
15. 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: Parallel and Distributed Systems (ICPADS), 2015 IEEE 21st International Conference on. pp. 682–691. IEEE (2015)
16. Zhao, Z.D., Shang, M.S.: User-based collaborative-filtering recommendation algorithms on hadoop. In: Knowledge Discovery and Data Mining, 2010. WKDD'10. Third International Conference on. pp. 478–481. IEEE (2010)
17. Zhou, Y., Wilkinson, D., Schreiber, R., Pan, R.: Large-scale parallel collaborative filtering for the netflix prize. In: International Conference on Algorithmic Applications in Management. pp. 337–348. Springer (2008)