A Large-Scale, Hybrid Approach for Recommending Pages Based on Previous User Click Pattern and Content

Mohammad Amir Sharif and Vijay V. Raghavan

The Center for Advanced Computer Studies
University of Louisiana at Lafayette
Lafayette, LA 70503, USA
{mas4108,vijay@cacs.louisiana.edu}

Abstract. In a large-scale recommendation setting, item-based collaborative filtering is preferable due to the availability of huge number of users' preference information and relative stability in item-item similarity. Item-based collaborative filtering only uses users' items preference information to predict recommendation for targeted users. This process may not always be effective, if the amount of preference information available is very small. For this kind of problem, item-content based similarity plays important role in addition to item co-occurrence-based similarity. In this paper we propose and evaluate a Map-Reduce based, large-scale, hybrid collaborative algorithm to incorporate both the content similarity and co-occurrence similarity. To generate recommendation for users having more or less preference information the relative weights of the item-item content-based and co-occurrence-based similarities are user-dependently tuned. Our experimental results on Yahoo! Front Page "Today Module User Click Log" dataset shows that we are able to get significant average precision improvement using the proposed method for user-dependent parametric incorporation of the two similarity metrics compared to other recent cited work.

Keywords: Recommender Systems, Item-based Collaborative Filtering, Map-Reduce, Item-Item content-based similarity, Item-Item co-occurrence-based similarity, Mahout.

1 Introduction

In this age of Internet, the amount of information we come across is overwhelming. It is really difficult to find relevant information useful for a person. There are numerous research studies in this branch of research called information filtering [1, 2, 3, 4]. Information filtering system assists users by filtering the data source and delivers relevant information to the users. When the delivered information comes in the form of suggestions, an information filtering system is called a recommender system.

Recommender systems are mainly classified as content-based and collaborative filtering-based. In content-based recommendation, user's profile-vector is matched with item's profile vector to generate recommendation [2]. Content-based system depends on well structured attributes and reasonable distribution of attributes across items. A content-based system is unlikely to find surprising connection. Rather, such a system aims to find a substituting item. In many cases, getting common attributes is not easy and complimentary items are preferred, rather than simple substitution. So, content-based systems are not preferred in many cases. In order to overcome these problems, collaborative filtering approach is introduced, which depends mainly on the users' item preferences information. It does not depend on the content information.

Collaborative filtering is classified as either user-based or item-based [3]. In the user-based collaborative filtering, recommendation is generated based on weighted average rating of similar or neighboring users' ratings. But as the number of users increases many users will not have sufficient rating and the user-item rating matrix will be very sparse. This can lead to finding no or dissimilar users as neighbors of a targeted user, making the recommendation task difficult. To overcome this sparsity problem, item-item collaborative recommendation is introduced, where item-item similarity is used to generate recommendation. Because it is more probable that each item will be rated by many users, finding effective neighbors of an item will be much easier.

In user-based method overcoming sparsity to get sufficient neighbor is acceptable, but in item-based collaborative algorithm, finding effective similar items is not enough, because some neighboring item may not be rated by the targeted users to contribute to the prediction calculation. The prediction is calculated by making weighted average of targeted user's rating to the neighboring items of the preferred item [4]. So if all or sufficient number of neighboring items can't contribute to recommendation calculation, then the recommendation will not be accurate. So, if a targeted user has a small number of preferred items, the recommendation generation needs to be modified.

Moreover, if the number of users and items increases to a massive scale, the traditional method of deploying an algorithm in a single machine does not work. So, scalability is a very common problem in current recommendation settings.

In this work, we propose to use a hybrid item-similarity score by combining item-item co-occurrence similarity and item-item content-based similarity. We use a parametric incorporation so that the parameter can have different values for different target users based on the length of their preference list. We test whether targeted users having a short preference lists get better recommendation if content-based similarity is given more weight for recommendation generation. Our experimental results validate the claim.

Moreover, we believe that the Map-Reduce based implementation of our proposed hybrid method is highly scalable to handle many recommendations in parallel. The experimental results on "Yahoo front page today module" dataset shows significant speed up in recommendation generation.

2 Literature Survey and Background

2.1 Sparsity Problem

In most e-commerce recommendation systems, the numbers of users and items are very large and many of the users don't rate or their actual preference is not obtainable.

In addition, even many popular items are not rated by many users. These factors cause user sparsity and item sparsity. Table 1 shows a user-item preference matrix where we can see that the matrix is not filled. In this case, the similarity calculation based on item cooccurrences will not be very much effective. Selecting either user-based or item-based collaborative approach automatically is also a method to handle sparsity [5]. Predictions corresponding to a user-based and item-based technique are calculated separately and, then, decisions are combined to make an integrated prediction in [5], which is infeasible for a large-scale implementation. Sometimes users sparsity is resolved by applying item-based filtering. But, in order to resolve item-sparsity, some studies have combined content-based item-similarity with the co-occurrence-based item similarity in item-based recommender systems [6, 7]. A weighted combination of content similarity and collaborative similarity is proposed in [6]. Table 2 shows an item-feature matrix. It is possible to calculate pair-wise similarity between items using the features of items. So, if the feature-based similarity and co-occurrence-based similarity are incorporated properly, better prediction is expected. Since the customization of weighting for the whole data set is practically infeasible, [7] proposes an incorporation technique giving the same weight for both the similarity values in multiplicative form. The performance of developed system in [7] is poor when the preference list size is small. Small preference list size hurts co-occurrence-based collaborative approach.

Table 1. User-Item preference matrix

Item User	I_1	I_2	I_3	I_4
U_1	$P_{1,1}$	$P_{1,2}$	$P_{1,3}$	?
U_2	$P_{2,1}$	$P_{2,2}$	$P_{2,3}$	$P_{2,4}$
U_3	$P_{3,1}$	$P_{3,2}$	?	$P_{3,4}$
U_4	?	$P_{4,2}$	?	?

So, if the active user does not have sufficient preference information, the resulting collaborative approach gives poor recommendation even though better item similarity is available. In this work we propose a user-dependent weighting technique so that, for the targeted users having short preference lists, more weight can be given on content-based similarity during prediction. On the other hand, users having longer preference lists will get more weight on co-occurrence based similarity prediction.

Table 2. Item-Feature matrix

Feature Item	F_1	F_2	F_3	F_4
I_1	$F_{1,1}$	$F_{1,2}$	$F_{1,3}$	$F_{1,4}$
I_2	$F_{2,1}$	$F_{2,2}$	$F_{2,3}$	$F_{2,4}$
I_3	$F_{3,1}$	$F_{3,2}$	$F_{3,3}$	$F_{3,4}$
I_4	$F_{4,1}$	$F_{4,2}$	$F_{4,3}$	$F_{4,4}$

2.2 Scalability Problem

As the number of users and items increases the computational complexity increases drastically in any recommendation setting.

In most of the recommender systems, the recommendation generation needs to be real time. There are already some works to do large-scale recommendation [8, 9]. While some of them have bottleneck in accuracy, others lack in the quality of performance. In this work, we propose an adaptive recommendation system that adjusts weight parameters according to the profile list lengths on top of the Apache Mahout's large-scale item-based recommendation systems. Mahout's Map-Reduce based recommendation systems run on top of Hadoop [10].

Map-Reduce. Map-reduce is a large-scale parallel computing framework developed by Google. A Map-reduce job mainly has two types of functions called *map ()* and *reduce()*, taking input from distributed file system (DFS).

The *Map* and *Reduce* functions of *Map-Reduce* are both defined with respect to data structured in (key, value) pairs. *Map* takes one pair of data with a type in one data domain, and returns a list of pairs in a different domain: `Map(k1,v1)→ list(k2,v2).`

The *Map* function is applied in parallel to every pair in the input dataset. This produces a list of pairs for each call. After that, the Map-Reduce framework collects all pairs with the same key from all lists and groups them together, creating one group for each key.

The *Reduce* function is then applied in parallel to each group, which in turn produces a collection of values in the same domain: `Reduce(k2, list(v2)) →list(k3, v3).`

Mahout's Large Scale Recommendation. In Mahout's algorithm the item similarity is calculated based on co-occurrences. Their distributed algorithm is based on parallel implementation of matrix multiplication. They run four Map-Reduce jobs to implement the recommendation.

Our content similarity-oriented, large-scale item based collaborative recommender system is built on top of an existing large-scale recommender system by apache Mahout [11]. We extended page co-occurrence to include content-based item-item similarity within the same Map-Reduce process. The structure is given in Fig. 1.

This algorithm runs four Map-Reduce jobs. The **first** Map-Reduce job builds user vector. Mapper takes file position as key and line of text containing user id, item id and preference as value; and outputs user id as key and (item id, preference) as value. The Reducer outputs user id as key and mahout's vector representation of items as value.

Second map-reduce job creates a co-occurrence matrix. The mapper takes user id as key and user vector as value; and outputs item id as key and other item id as value. The output from the mapper is itemId as key; and another-item with corresponding similarity as value. The reducer here outputs item id as key and a column vector of co-occurrence matrix as value.

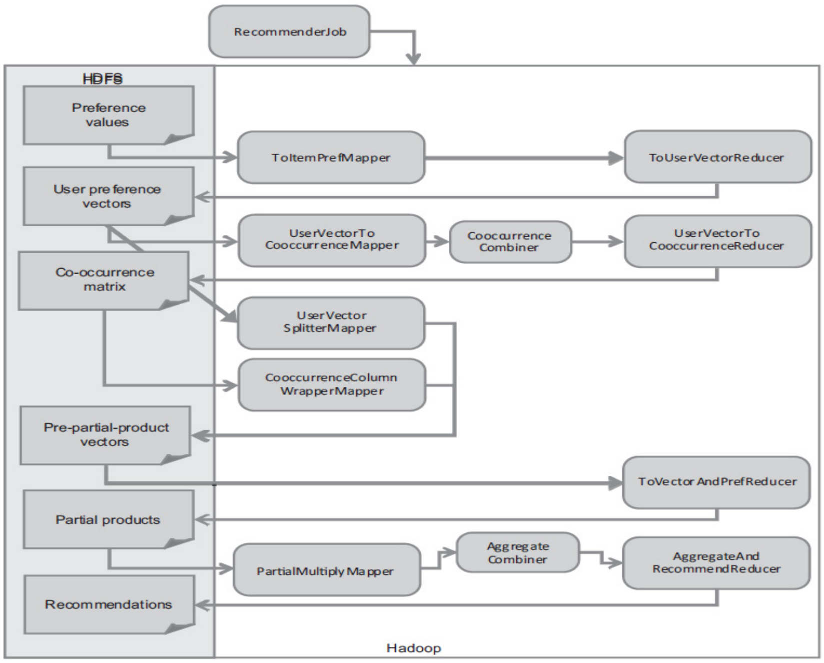

Fig. 1. Architecture of Mahout's Recommendation Engine

Inside-out technique is used for map-reduce based matrix multiplication. In **third** map-reduce job one mapper called *PartialMultiply1* takes the output of the first map-reduce job as input where user id is key and user's preference vector is the value. The output of this map job is item id as key and (user id, preference value) as value. Another mapper called *Partial Multiply2* inputs and outputs item id as key and co-occurrence matrix column Vector as value. The *Partial multiply* reducer inputs item id as key; and many (user ID, preference) pairs with a co-occurrence matrix column vector as value, which are value parts from previous two mapper's output. The reducer outputs item id as key and column vector with (user ID, preference) pair as value.

In the **fourth** map-reduce job a mapper called *aggregate mapper* takes the key value pair as input from the output of previous map-reduce job i.e. item id as key and column vector and (user ID, preference) pair as value. And the mapper outputs user id as key and column vector times preference as value. *Aggregate reducer* receives user id as key and vectors from previous mapper output for corresponding user id. Reducer sums to make recommendation vector and finds top n values for recommendations. For the top n values, the reducer outputs user id as key and (item ID, value) as value.

3 Our Approach

3.1 Content-Based Item-Item Similarity Calculation

As the content-based similarity is stable, we calculated the item-item content-based similarity off-line, and then incorporated it with the co-occurrence similarity later.

We used Mahout's distributed *row similarity* calculation module for this content-based similarity calculation. Each item is represented as a vector of features. Then cosine similarity measure is used to calculate similarity among two items, which gives a similarity value between 0 to 1.

3.2 Computation

We merged two similarity matrices in the *user vector to co-occurrence-mapper* and *co-occurrence-reducer* Map-Reduce job shown in Fig. 1. This Map-Reduce job takes pair wise similarity value of each pair of items. We inputted two kinds of similarity values for each pair of items from file; one co-occurrence-based and other content-based. The output of the mapper is item ID as *key* and another item, with corresponding similarity values, as *value*. So after all the map tasks are completed, each item will be emitted twice for same item with different similarity values. These key-value pairs are sent to the *co-occurrence-reducer.* The input to reducer is item ID as *key* and a vector of rest of the items with corresponding similarity values to the key item as *value*. In this vector there will be two entries of same item with two different kinds of similarities. We used a linked-list to find those duplicate items, and the corresponding similarities are summed-up to get an aggregated similarity among items. At the end, the reducer emits item ID as key and the corresponding updated similarity vector in terms of other items as key. In this part, we just merge the two similarity matrices into one. As the co-occurrence similarity ranges from [0, 185] and content similarity ranges from (0, 1], it is very easy to separate these similarity components later and give different weights of importance for combining the two similarity values based on different user preference list length, during recommendation generation.

In *aggregateAndReccomendReducer* module, we apply separate weights to the different similarity components based on the users' preference list length. In order to do so, we separate the whole and fractional part of the similarity value. The whole number is co-occurrence similarity and normalized to [0, 1] scale, the fractional part is content-similarity. After normalization we compute the weighted sum of the two different similarities for a user, based on her preference list length. In this way, the two different similarities are merged to generate the prediction.

3.3 Incorporating Content-Based and Co-occurrence-based Similarity

In our similarity calculation method, we used the following equation to calculate similarity, $Sim(ItemA, ItemB, U)$ among two items for a user U,

$$\begin{aligned} Sim(ItemA, ItemB, U) \\ &= F(U) \times Content_{Sim(ItemA,ItemB)} + \left(1 - F(U)\right) \\ &\times Cooccurrence_{sim(ItemA,ItemB)} \end{aligned}$$

$$where, F(U){:}\ Content\ similarity\ weight\ to\ be\ considered\ for\ user\ U$$
$$1 - F(U){:}\ Cooccurrence\ similarity\ weight\ to\ be\ considered\ for\ user\ U$$
$$Content_{Sim(ItemA,ItemB)}{:}\ Content\ similarity\ among\ items\ ItemA\ and\ ItemB$$
$$Cooccurrence_{sim(ItemA,ItemB)}{:}\ Cooccurrence\ similarity\ of\ ItemA\ and\ ItemB$$

4 Experimental Setup

We used the Yahoo! Front Page Today Module User Click Log Dataset for the experiments. The dataset is a fraction of user click log for news articles displayed in the Featured Tab of the Today Module on Yahoo! Front Page (http://www.yahoo.com) during the first ten days in May 2009. The dataset contains 45,811,883 user visits to the Today Module. For each visit, both the user and each of the candidate articles are associated with a feature vector of dimension 6. These features were represented by some numerical numbers without any identification. We preprocessed the data set to extract article click information for each user and the article vectors for each article for this experiment. There were 271 articles that were displayed in Yahoo! front page in those ten days.

4.1 Preference List Length Based Evaluation

We choose several different sets of users based on different preference list lengths for our evaluation. In this case, the preference means clicked pages. Each selected user set had users having preference list length 6 to 10, 11 to 20, 21 to 35, 36 to 55, more than 35 and more than 55. Although we had around 1.3 million users, very few of them had prefernce lists of size greater than 5. There were 560 users having preference list length 6 to 10, 333 users with length 11 to 20, 142 users with length 21 to 35, 87 users with length 36 to 55 and 214 users having prefernce length more than 55. For these sets, we trained the recommender system by 70% of the preferences of the users in the current user set along with 100% of other users' preferences and kept the remaining 30% preference list in the current user set for testing i.e. testing preferences. For these users, we explore the recommendations generated by the system for different weight combinations of content similarity and co-occurrence similarity. For each selected user set, we ran the experiment by increasing the content similarity weight; that is, by taking the content-based similarity and co-occurrence-based similarity weight pair respectively as (0, 1), (0.1, 0.9), (0.2, 0.8), (0.3, 0.7), (0.4, 0.6), (0.6, 0.4), (0.7, 0.3), (0.8, 0.2), (1, 0). Considering the rank order of first 10 recommendations, we calculated average precision for an active user, for each combination of weights, based on ranking of pages in the testing preferences. Then we found the mean average precision (MAP) over all the users within those selected user sets [12].

Fig. 2, 3 and 4 show how the recommender's performance changes with the increase of content similarity weight or decrease of co-occurrence similarity weight for users having preference list length respectively 6 to 10, 11 to 20, 21 to 35 and 36 to 55, more than 35 and more than 55.

From Fig. 2 and 3, we can see that for users having prefernce list length 6 to 10, 11 to 20 and 21 to 35 the best performance is obtained for content similarity weight 1.0 and cooccurence similarity weight 0.0. For users with prefence list length 36 to 55, we get better performance with content similarity weight 0.4 and cooccurence similarity weight 0.6, which shows the imporatnce of co-occurrence as the length grows. Fig. 4 says that for users having prefernce list length greater than 36 or 55, we get better

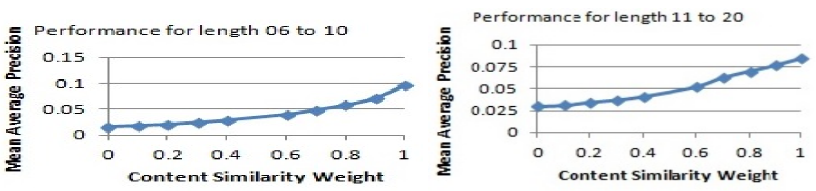

Fig. 2. Change of performance with increasing content similarity weight or decreasing co-occurrence similarity weight for users having preference list length 06 to 10 and 11 to 20

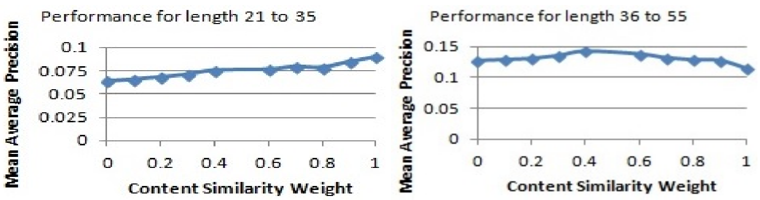

Fig. 3. Change of performance with increasing content similarity weight or decreasing co-occurrence similarity weight for users having preference list length of 21 to 35 and 36 to 55

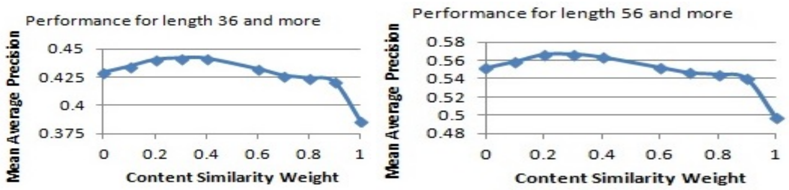

Fig. 4. Change of performance with increasing content similarity weight or decreasing co-occurrence similarity weight for users having preference length greater than 35 and 55

performance at content similarity weight of 0.4 and 0.3 respetcively. It also shows the importance of co-occurrence weight for users having long prefernce list. So, for shorter preference list length, less than 36, we give more weight on content.

Observing the results in Fig. 2, 3 and 4, we selected content similarity weight 1.0 for users having length less than 36; content similarity weight 0.4 for length 36 to 55 and content similarity weight 0.3 for length greater than 55. We used this weighting mechanism to do our full-scale testing. We tested our system with all the 1336 users having prfernce length greater than 5. Again, for each user in this set of 1336 users, 70% of papges in the preference list are used for training and the rest for testing. Table 3 shows the performance comparison of our user-dependent method with systems having only co-occurrence similarity, content similarity; and system described by Puntheeranurak, S., where the two similarities are just multiplied without using any parameter [7]. We can see from Table 3 that, we got 22% better performance than just using a simple merge process without parameter tuning.

Table 3. Comparative performance of our user-dependent parametric method with alternative approaches for all the users having preference list length greater than 5

	Co-occurrence similarity only	Content similarity only	Method in reference[7]	User-dependent adjustment
MAP	0.1174517	0.1578064	0.1253729	0.16152156

4.2 Scalability for Online Recommendations to Growing Number of Users

We ran our recommendation algorithm both in single node Hadoop ecosystem in a virtual machine and in a Cloudera 4.5 cluster with 3 nodes with one master node and two slave nodes. Each machine was with quad-core CPU with speed 2.4GHz, 4GB memory, running on Centos 6.2 OS. Table 4 shows the comparative elapsed time requirement for recommendation generation to varying number of users. The time required to generate a similarity matrix, which is used in recommendation generation is approximately 11 minutes for single machine and 5.5 minutes for the cluster environment. In table 4, we report the required time to generate recommendation from already created similarity matrix for different number of users.

Table 4. Comparison of recommendation time for different number of users

No. of users	Single Machine(time)	Cluster(time)
100K	3m29sec	1m32sec
200K	4m15sec	1m44sec
300K	5m1sec	1m57sec
400K	6m28sec	2m5sec
500K	7m52sec	2m26sec

From Table 4, we can see that for 100,000 users to 500,000 users the running time increases linearly with the number of targeted users for the single node recommendation generation. In the distributed environment, on a Hadoop cluster, the running time is also linear, but increases at a lower rate. Definitely a cluster with many nodes could make the running time nearly constant.

5 Conclusion

In this paper we have presented a Map-Reduce based, scalable recommender system that uses a unique, preference list length based weighting technique to incorporate content-based and co-occurrence-based item-item similarities to help make better recommendations in the situation where preference list sizes are small. Our length-based weighting technique gives 22% better performance compared to a recent work described in [7]. As we selected only users having preference list lengths greater than five for each test set, we could not show full potential of length-based approach for determining the weight parameters. But, in future, we plan to use the whole dataset to determine the length-based parameters applicable to a more global context. We will also use a bigger compute cluster to demonstrate better speed-up.

References

1. Hanani, U., Shapira, B., Shoval, P.: Information Filtering: Overview of Issues, Research and Systems. User Modeling and User-Adapted Interaction 11, 203–259 (2001)
2. Delgado, J., Ishii, N., Ura, T.: Content-based Collaborative Information Filtering: Actively Learning to Classify and Recommend Documents. In: Klusch, M., Weiss, G. (eds.) CIA 1998. LNCS (LNAI), vol. 1435, pp. 206–215. Springer, Heidelberg (1998)
3. Adomavicius, G., Tuzhilin, A.: Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-art and Possible Extensions. IEEE Transactions on Knowledge and Data Engineering 11(6), 734–749 (2005)
4. Su, X., Khoshgoftaar, T.M.: A survey of collaborative filtering techniques. Journal of Advances in Artificial Intelligence (2009)
5. Hu, R., Lu, Y.: A Hybrid User and Item-based Collaborative Filtering with Smoothing on Sparse Data. In: Proceedings of the 16th International Conference on Artificial Reality and Telexistence–Workshops (2006)
6. Gong, S.J., Ye, H.W., Shi, X.Y.: A Collaborative Recommender Combining Item Rating Similarity and Item Attribute Similarity. International Seminar on Business and Information Management (2008)
7. Puntheeranurak, S., Chaiwitooanukool, T.: An Item-based Collaborative Filtering Method using Item-based Hybrid Similarity. In: 2nd International Conference on Software Engineering and Service Science (2011)
8. Jiang, J., Lu, J., Zhang, G., Long, G.: Scaling-up Item-based Collaborative Filtering Recommendation Algorithm based on Hadoop. IEEE World Congress on Services (2011)
9. Chen, Y., Pavlov, Y.: Large Scale Behavioral Targeting. In: 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2009)
10. Apache Hadoop, `http://hadoop.apache.org/`
11. Apache mahout, `https://mahout.apache.org/`
12. Text retrieval quality, `http://www.oracle.com/technetwork/database/enterprise-edition/imt-quality-092464.html`