Group Recommendations in MapReduce

Vasilis Efthymiou, Petros Zervoudakis, Kostas Stefanidis, and Dimitris Plexousakis

Abstract Recommender systems have received significant attention, with most of the proposed methods focusing on recommendations for single users. However, there are contexts in which the items to be suggested are not intended for a user but for a group of people. For example, assume a group of friends or a family that is planning to watch a movie or visit a restaurant. In this paper, we propose an extensive model for group recommendations that exploits recommendations for items that similar users to the group members liked in the past. We follow two different approaches for offering recommendations to the members of a group: considering the members of a group as a single user, and recommending to this user items that similar users liked, or estimating first how much each group member would like an item, and then, recommend the items that would (dis)satisfy the most (least) members of the group. For each of the two approaches, we introduce a different MapReduce algorithm, and evaluate the results in real data from the movie industry.

1 Introduction

Efficient top-k processing is a crucial requirement in many interactive environments, such as the Web, that involve huge amounts of data (for surveys on top-k computations, see [4, 12]). As an instance, consider that with the growing complexity of the

Vasilis Efthymiou
University of Crete, Greece, e-mail: vefthym@csd.uoc.gr

Petros Zervoudakis
University of Crete, Greece, e-mail: zervoudak@csd.uoc.gr

Kostas Stefanidis
University of Tampere, Finland, e-mail: kostas.stefanidis@uta.fi

Dimitris Plexousakis
ICS-FORTH, Greece, e-mail: dp@ics.forth.gr

Web [2], users often find themselves overwhelmed by the mass of choices available. To facilitate users in their selection process, recommender systems offer suggestions of potential interest on items [9]. In particular, recommender systems aim at providing recommendations to users or groups of users by estimating their preferences and recommending those items featuring the maximal predicted preference. A prerequisite for determining such recommendations is historical data on the users' interests, e.g., their purchase history.

Collaborative filtering [11] is a widely used recommendations technique, in which the key concept is to locate users with similar rating behavior to the query user; the preferences of these users are employed then for issuing recommendations for the query user. The simplest, naïve approach for finding similar users is by linearly scanning the whole user base. Clearly, this is a costly process for large recommender systems with millions of users. More efficient approaches for identifying users sharing similar preferences build user models and employ these models for rating prediction. User models might be derived through clustering users into groups of similar users (e.g., [7, 8]).

Based on recent studies [8], although a clustering approach is much faster than the naïve approach, the naïve is superior in terms of quality. Luckily, nowadays, even the naïve approach of linearly scanning the database to locate similar users might be implemented efficiently through distributed computing, allowing exact solutions. In this paper, we propose a distributed approach for computing recommendations using MapReduce. Specifically, we consider group recommendations (e.g., [10, 7]), i.e., recommendations for groups of users instead of single users.

2 Group Recommendations Model

Assume a recommender system, where I is the set of items to be rated and U is the set of users in the system. A user $u \in U$ might rate an item $i \in I$ with a score $rating(u,i)$ in $[0,1]$; let R be the set of all ratings recorded in the system. Typically, the cardinality of the item set I is high and users rate only a few items. The subset of users that rated an item $i \in I$ is denoted by $U(i)$, whereas the subset of items rated by a user $u \in U$ is denoted by $I(u)$.

For the items unrated by the users, recommender systems estimate a relevance score, denoted as $relevance(u,i)$, $u \in U$, $i \in I$. There are different ways to estimate the relevance score of an item for a user. In the content-based approach (e.g., [6]), the estimation of the rating of an item is based on the ratings that the user has assigned to similar items, whereas in collaborative filtering systems (e.g., [5]), this rating is predicted using previous ratings of the item by similar users. In this work, we follow the collaborative filtering approach. Similar users are located via a *similarity function* $simU(u,u')$ that evaluates the proximity between $u,u' \in U$.

We define the distance between two users $u,u' \in U$ as the Euclidean distance over the items rated by both:

$$distU(u,u') = \left(\sqrt{\sum_{i \in \mathscr{I}(u) \cap \mathscr{I}(u')} (rating(u,i) - rating(u',i))^2} \right) / |\mathscr{I}(u) \cap \mathscr{I}(u')|. \quad (1)$$

Then, $simU(u,u') = 1 - distU(u,u')$.

We use F_u to denote the set of the most similar users to u, hereafter referred to as the *friends* of u.

Definition 1. Let $\mathscr{U}$ be a set of users. The friends $\mathscr{F}_u \subseteq \mathscr{U}$ of a user $u \in \mathscr{U}$ consist of all those users $u' \in \mathscr{U}$ that are similar to u with respect to a similarity threshold δ, i.e., $\mathscr{F}_u = \{u' \in \mathscr{U} : simU(u,u') \geq \delta\}$.

Clearly, one could argue for other ways of selecting $\mathscr{F}_u$, e.g., by selecting the k most similar users to u. Our main motivation is that we opt for selecting only highly connected users. Given a user u and his friends $\mathscr{F}_u$, if u has expressed no preference for an item i, the relevance of i for u is estimated as:

$$relevance_{\mathscr{F}_u}(u,i) = \frac{\sum_{u' \in (\mathscr{F}_u \cap U(i))} simU(u,u') rating(u',i)}{\sum_{u' \in (\mathscr{F}_u \cap U(i))} simU(u,u')}. \quad (2)$$

After estimating the relevance scores of all unrated user items, the top-k rated items are recommended to the user.

Since recommendations are typically personalized, different users receive different suggestions. However, there are cases where a group of people participates in a single activity. For instance, visiting a restaurant or a tourist attraction, watching a movie or a TV program, and selecting a holiday destination are examples of recommendations well suited for groups of people. For this reason, recent studies (e.g., [10, 1, 7]) have focused on group recommendations, trying to satisfy the preferences of all the group members.

In general, we consider two different ways to produce recommendations for groups: (i) the *multi-users group* approach, and (ii) the *single-user group* approach. In the multi-users group approach, we first estimate the relevance scores of the unrated items for each user in the group, and then, aggregate these predictions to compute the suggestions for the group. Specifically, let $\mathscr{U}$ be a set of users and $\mathscr{I}$ be a set of items. Then, given a group of users $\mathscr{G} \subseteq \mathscr{U}$, the group relevance of an item $i \in \mathscr{I}$ for $\mathscr{G}$, such that, $\forall u \in \mathscr{G}$, $i \notin I(u)$, is:

$$relevance^m(\mathscr{G},i) = Aggr_{u \in \mathscr{G}}(relevance_{\mathscr{F}_u}(u,i)). \quad (3)$$

As in [7], we consider three different designs regarding the aggregation method $Aggr$: (i) the *least misery design*, capturing cases where strong user preferences act as a veto (e.g., do not recommend steakhouses to a group when a member is vegetarian), (ii) the *fair design*, capturing more democratic cases where the majority of the group members is satisfied, and (iii) the *most optimistic design*, capturing cases where the most satisfied member of the group acts as the most influential one (e.g., recommend a movie to a group when a member is highly interested in it and the

rest have reasonable satisfaction). In the least misery (resp., most optimistic) design, the predicted relevance score of an item for the group is equal to the minimum (resp., maximum) relevance score of the item scores of the members of the group, while the fair design, that assumes equal importance among all group members, returns the average score.

In the single-user group approach, intuitively, we consider the group as a single user, search for the friends of this user, and based on their ratings, compute the group recommendations. This way, $\mathscr{F}_{\mathscr{G}} = \{u' \in \mathscr{U} : sim\mathscr{G}(\mathscr{G},u') \geq \delta\}$, where $sim\mathscr{G}(\mathscr{G},u') = f_{u\in\mathscr{G}}(simU(u,u'))$ and f is an aggregated function, e.g., the *minimum* or *average* function. Then, as above:

$$relevance^s(\mathscr{G},i) = \frac{\sum_{u'\in(\mathscr{F}_{\mathscr{G}}\cap U(i))} sim\mathscr{G}(\mathscr{G},u')rating(u',i)}{\sum_{u'\in(\mathscr{F}_{\mathscr{G}}\cap U(i))} sim\mathscr{G}(\mathscr{G},u')}. \quad (4)$$

3 Implementation in MapReduce

In this section, we present the implementation of our recommender system in MapReduce[1]. First, we describe the implementation of the multi-user approach, and then we continue with the single-user approach. In both cases, we assume that our input consists of a set of user rating triples $R = \{(u,i,rating(u,i))|u \in U, i \in I\}$, and a set of user ids $\mathscr{G} \subseteq U$, composing the group of interest. We conclude this section with a set of preliminary experiments, comparing the two approaches.

3.1 Multi-user group approach

In the multi-user approach, we compute the list of friends for each member of the group (Definition 1), the relevance of every item to each member of the group (Equation 2), and, finally, aggregate those scores to get the final relevance of each item for the group (Equation 3). In order to identify the friends of each member, we compute the similarity between each member of the group and every other user, outside the group. The implementation of this approach consists of three MapReduce jobs.

Job 1 - Partial distances & unrated items. We group the input ratings by item id in the map phase, emitting $(i, < u, rating(u,i) >)$ pairs. The reduce phase produces two different outputs. If one (or more) member(s) of the group $\mathscr{G}$ is (are) among the users who have rated an item i, then, we compute a so-called *partial* distance between each member u and each non-member u' who have rated this item $part(u,u') = (rating(u,i) - rating(u',i))^2$, and emit a pair of the form $(< u,u' >, part(u,u'))$. We will need this partial distance later, to compute the value of Equation 1. If none of the members of the group $\mathscr{G}$ have rated an item i, then i

[1] Source code available at: https://github.com/vefthym/GroupRecsMR

is one of the candidate items for recommendation, and we simply emit a pair of the form $(i, < u, rating(u,i) >)$ in a different path than the previous output.

Job 2 - User similarities. This job forwards the partial distances from Job 1 to the reducers, grouping those distances by pairs of users. The reducers then, having all the partial distances for each user pair $u, u' \in U$, where $u \in \mathscr{G}$, compute the value of Equation 1 and emit a pair $(< u, u' >, simU(u,u'))$. Based on Definition 1, we can filter the emitted results to only those whose similarity value is above a certain threshold δ, i.e., emit only the similarity of u to each of his/her friends $u' \in \mathscr{F}_u$.

Job 3 - Final relevance. The final job forwards a list of unrated items and the ratings of those items from the non-members of $\mathscr{G}$ from Job 1, grouped by item id, to the reducers. To reduce the network traffic, the emitted ratings are filtered to keep only those given by friends of a group member, i.e, from $u' \in \bigcup_{u \in \mathscr{G}} \mathscr{F}_u$. Having the similarities of group members to their friends from Job 2 loaded in memory, the reducers then compute the relevance score of each unrated item for every group member (Equation 2), and aggregate those scores to produce the final group relevance values of Equation 3. The final sorting and top-k selection of those relevance values is trivial when k elements are small enough to fit in memory. When this is not the case, we can use the top-k MapReduce algorithm suggested in [3].

3.2 Single-user group approach

The MapReduce algorithm for the single-user group approach is similar to the multi-user approach. This time, four MapReduce jobs are required, as explained next.

Job 1 - Partial distances & unrated items. Identical to the multi-user approach.

Job 2 - User similarities. The only difference to Job 2 of the multi-user approach, is that we cannot filter the results, based on the given similarity threshold, as an aggregated similarity is required to identify the friends of the group.

Job 3 - Friends of the group. This job groups the similarity results of Job 2 by non-member user, to get, in the reduce phase, only the users with an aggregated similarity score above the similarity threshold.

Job 4 - Final relevance. Same as the multi-user approach, without the aggregation.

3.3 Comparison

To compare the two approaches, we present our experimental results over the MovieLens datasets[2], with 100K and 1M movie ratings. Our experiments were performed on a cluster of 15 virtual machines, each having 8 CPUs and 8 GB of RAM.

[2] http://grouplens.org/datasets/movielens/

The execution times are presented in Figure 1, showing that the multi-user group model is faster than the single-user group model, due to having one MapReduce job less. We plan to evaluate our approach on larger datasets and evaluate the quality of the recommendations with user studies.

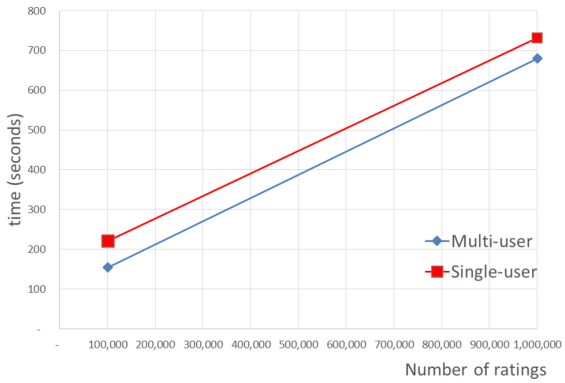

Fig. 1 Execution times (in seconds) of the two approaches over 100K and 1M ratings.

References

1. L. Baltrunas, T. Makcinskas, and F. Ricci. Group recommendations with rank aggregation and collaborative filtering. In *RecSys*, pages 119–126, 2010.
2. V. Christophides, V. Efthymiou, and K. Stefanidis. *Entity Resolution in the Web of Data*. Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool, 2015.
3. V. Efthymiou, K. Stefanidis, and E. Ntoutsi. Top-k computations in mapreduce: A case study on recommendations. In *IEEE Big Data*, pages 2820–2822, 2015.
4. I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-*k* query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4), 2008.
5. J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl. Grouplens: Applying collaborative filtering to usenet news. *Commun. ACM*, 40(3):77–87, 1997.
6. R. J. Mooney and L. Roy. Content-based book recommending using learning for text categorization. In *ACM DL*, pages 195–204, 2000.
7. E. Ntoutsi, K. Stefanidis, K. Nørvåg, and H. Kriegel. Fast Group Recommendations by Applying User Clustering. In *ER*, pages 126–140, 2012.
8. E. Ntoutsi, K. Stefanidis, K. Rausch, and H. Kriegel. "Strength Lies in Differences": Diversifying Friends for Recommendations through Subspace Clustering. In *CIKM*, pages 729–738, 2014.
9. F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors. *Recommender Systems Handbook*. Springer, 2011.
10. S. B. Roy, S. Amer-Yahia, A. Chawla, G. Das, and C. Yu. Space efficiency in group recommendation. *VLDB J.*, 19(6):877–900, 2010.
11. J. J. Sandvig, B. Mobasher, and R. D. Burke. A survey of collaborative recommendation and the robustness of model-based algorithms. *IEEE Data Eng. Bull.*, 31(2):3–13, 2008.
12. K. Stefanidis, G. Koutrika, and E. Pitoura. A survey on representation, composition and application of preferences in database systems. *ACM Trans. Database Syst.*, 36(3):19, 2011.