gRecs: A Group Recommendation System based on User Clustering

Irene Ntoutsi[1], Kostas Stefanidis[2⋆], Kjetil Norvag[2], and Hans-Peter Kriegel[1]

[1] Institute for Informatics, Ludwig Maximilian University, Munich
{ntoutsi, kriegel}@dbs.ifi.lmu.de
[2] Department of Computer and Information Science, Norwegian University of Science and Technology, Trondheim
{kstef,Kjetil.Norvag}@idi.ntnu.no

Abstract. In this demonstration paper, we present *gRecs*, a system for group recommendations that follows a collaborative strategy. We enhance recommendations with the notion of support to model the confidence of the recommendations. Moreover, we propose partitioning users into clusters of similar ones. This way, recommendations for users are produced with respect to the preferences of their cluster members without extensively searching for similar users in the whole user base. Finally, we leverage the power of a top-k algorithm for locating the top-k group recommendations.

1 Introduction

Recommendation systems provide suggestions to users about movies, videos, restaurants, hotels and other items. The large majority of recommendation systems are designed to make personal recommendations, i.e., recommendations for individual users. However, there are cases in which the items to be suggested are not intended for personal usage but for a group of users. For example, a group of friends is planning to watch a movie or to visit a restaurant. For this reason some recent works have addressed the problem of identifying recommendations for a group of users, trying to satisfy the preferences of all the group members.

Our work falls into the collaborative filtering approach, i.e., we offer user recommendations for items that similar users liked in the past. We introduce the notion of support in recommendations to model how confident the recommendation of an item for a user is. We also apply user clustering for organizing users into clusters of users with similar preferences. We propose the use of these clusters to efficiently locate similar users to a given one; this way, searching for similar users is restricted within his/her corresponding cluster instead of the whole database. Moreover, we exploit a top-k algorithm to efficiently identify the k most prominent items for the group.

⋆ This work was carried out during the tenure of an ERCIM "Alain Bensoussan" Fellowship Programme. This Programme is supported by the Marie Curie Co-funding of Regional, National and International Programmes (COFUND) of the European Commission.

The rest of this demostration paper is organized as follows: The *gRecs* framework is introduced in Section 2. The system architecture is presented in Section 3. Section 4 describes the *gRecs* demonstration scenario for group recommendations in a movies application.

2 The *gRecs* Group Recommendations Framework

Assume a set of items $\mathcal{I}$ and a set of users $\mathcal{U}$ interacting with a recommendation application. Each user $u \in \mathcal{U}$ may express a preference for an item $i \in \mathcal{I}$, which is denoted by $preference(u, i)$ and lies in the range $[0.0 - 1.0]$. For the items unrated by the users, we estimate a *relevance score*, denoted as $relevance(u, i)$, where $u \in \mathcal{U}$, $i \in \mathcal{I}$. To do this, a recommendation strategy is invoked.

We distinguish between personal recommendations referring to a single user and group recommendations referring to a set of users.

Personal recommendations. There are different ways to estimate the relevance of an item for a user. Our work follows the collaborative filtering approach, such as [5] and [3]. To produce relevance scores for unrated items for a user, we employ preferences of users that exhibit similar behavior to the given user. Similar users are located via a *similarity function* $simU(u, u')$, that evaluates the proximity between u and u'. We use $\mathcal{F}_u$ to denote the set of the most similar users to u. We refer to such users as the *friends* of u. Several methods can be employed for selecting $\mathcal{F}_u$. A straightforward method is to locate the users u' with $simU(u, u')$ greater than a threshold δ. This is the method used here.

Given a user u and his friends $\mathcal{F}_u$, if u has not expressed any preference for an item i, the relevance of i for u is commonly estimated as follows:

$$relevance(u, i) = \frac{\sum_{u' \in f_u \wedge \exists preference(u', i)} simU(u, u') \times preference(u', i)}{\sum_{u' \in f_u \wedge \exists preference(u', i)} simU(u, u')}.$$

Typically, users rate only a few items in a recommendation application because of the huge amount of the available items. This is our motivation for introducing the notion of *support* for each suggested item i, for user u. Support defines the percentage of friends of u that have expressed preferences for i, that is, $support(u, i) = \frac{|S, S \subseteq \mathcal{F}_u, s.t. \forall u' \in S, \exists preference(u', i)|}{|\mathcal{F}_u|}$.

To estimate the worthiness of an item recommendation for a user, we propose to combine the *relevance* and *support* scores in terms of a *value* function. Formally, the *personal value* of an item $i \in \mathcal{I}$ for a user $u \in \mathcal{U}$, such that, $\nexists\ preference(u, i)$, is defined as: $value(u, i) = w_1 \times relevance(u, i) + w_2 \times support(u, i)$, $w_1 + w_2 = 1$.

Group recommendations. In addition to personal recommendations, there are contexts in which people operate in groups, and so, a model for group recommendations should be defined. Some approaches have been recently proposed towards this direction (e.g., [2]).

In general, collaborative filtering combines the preferences of the single users to predict the preferences for the group as a whole. To this end, in our approach,

we first compute the *personal value* scores for the unrated items for each user of the group. Based on these predictions, we then produce the aggregated value scores for the group. Formally, given a group of users $\mathcal{G}$, $\mathcal{G} \subseteq \mathcal{U}$, the group value of an item $i \in \mathcal{I}$ for $\mathcal{G}$, such that, $\forall u \in \mathcal{G}$, $\nexists\, preference(u, i)$, is: $value(\mathcal{G}, i) = Aggr_{u \in \mathcal{G}}(value(u, i))$.

We employ three different designs regarding the aggregation method $Aggr$, each one carrying different semantics: (i) the *least misery design*, capturing cases where strong user preferences act as a veto, (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 member. In the least misery (resp., most optimistic) design the predicted value score for the group is equal to the minimum (resp., maximum) value score of the scores of the members of the group, while the fair design returns the average score.

Given a group of users and a restriction k on the number of the recommended items, our goal is to provide the group with k suggestions for items that are highly valued, without computing the group value scores of all database items.

3 *gRecs* System Overview

In this section, we describe the main components of the architecture of our system. A high level representation is depicted in Figure 1. Given a group of users, we first locate the friends of each user in the group. Friends preferences are employed for estimating personal recommendations, while in turn, personal recommendations are aggregated into recommendations for the whole group.

Friends generator. This component takes as input a group of users $\mathcal{G}$ and returns the friends $\mathcal{F}_u$ of each user u in the group. The naive approach for finding the friends of all users in $\mathcal{G}$ requires the online computation of all similarity values between each user in $\mathcal{G}$ and each user in $\mathcal{U}$. We compute the similarity between two users with regard to their Euclidean distance. This however, is too expensive for a real-time recommendation application where the response time is an important aspect for the end users. To speed up the recommendation process, we perform preprocessing steps offline. More specifically, we organize users into clusters of users with similar preferences. For partitioning users into clusters, we use a hierarchical agglomerative clustering algorithm that follows a bottom-up strategy. Initially, the algorithm places each user in a cluster of his own. Then, at each step, it merges the two clusters with the greatest similarity. The similarity between two clusters is defined as the minimum similarity between any two users that belong to these clusters. The algorithm terminates when the clusters with the greatest similarity, have similarity smaller than δ. In this clustering approach, we consider as friends of each user u the members of the cluster that u belongs to. This set of users is a subset of $\mathcal{F}_u$.

Personal recommendations generator. In this step, we estimate the personal value scores of each item for each user in $\mathcal{G}$. To perform this operation, we

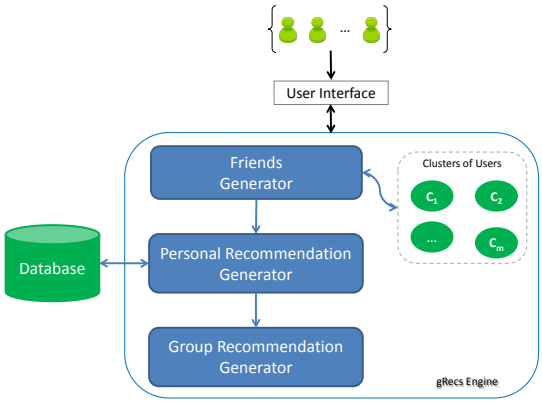

Fig. 1. *gRecs* system architecture.

employ the outputs of the previous step, i.e., the friends of the users in $\mathcal{G}$. Given a user $u \in \mathcal{G}$ and his friends $\mathcal{F}_u$, the procedure for estimating the $value(u, i)$ of each item i in $\mathcal{I}$ requires the computation of $relevance(u, i)$ and $support(u, i)$. Pairs of the form $(i, value(u, i))$ are maintained in a set $\mathcal{V}_u$. This component is also responsible for ranking, in descending order, all pairs in $\mathcal{V}_u$ on the basis of their personal value score.

Group recommendations generator. This component generates the k highest group-valued item recommendations for the group of users $\mathcal{G}$. To do this, we combine the personal value scores computed from the previous step by using either the *least misery*, the *fair* or the *most optimistic* design. Instead of following the common way of computing the group value scores of all items and ranking the items based on these scores, we employ the TA algorithm [4] for efficient top-k computation. Note that TA is correct when the group value scores of the items are obtained by combining their individual scores using a monotone function. In our approach, aggregations are performed in a monotonic fashion, hence the applicability of the algorithm is straightforward.

4 Demonstration

The *gRecs* system for group recommendations has been implemented in JAVA on top of MySQL. We demonstrate our method using a movie ratings database [1]. In particular, we form groups of users with different semantics and choose an aggregation design from the available ones. After estimating the top-k group value scores, users are presented with the recommended movies. An explanation is also provided along with each movie, i.e., why this specific recommendation appears in the top-k list. For the *least misery* (resp., *most optimistic*) design, we report with each movie its group value score and the member of the group

with the minimum (resp., maximum) personal value score for the movie, i.e., the member that is responsible for this selection. Similarly, for the *fair* design, we report with each movie the members of the group with personal value scores for the movie close to the group value score of the movie, i.e., the members that are highly satisfied, and hence, direct towards this selection.

References

1. Movielens data sets. Available online at: http://www.grouplens.org/node/12; visited on Oct. 2011.
2. S. Amer-Yahia, S. B. Roy, A. Chawla, G. Das, and C. Yu. Group recommendation: Semantics and efficiency. *PVLDB*, 2(1):754–765, 2009.
3. J. S. Breese, D. Heckerman, and C. M. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *UAI*, pages 43–52, 1998.
4. R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.
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.