DiRec: Diversified Recommendations for Semantic-less Collaborative Filtering

Rubi Boim, Tova Milo, Slava Novgorodov

School of Computer Science
Tel-Aviv University
{boim,milo,slavanov}@post.tau.ac.il

***Abstract*—In this demo we present DiRec , a plug-in that allows Collaborative Filtering (CF) Recommender systems to diversify the recommendations that they present to users. DiRec estimates items diversity by comparing the rankings that different users gave to the items, thereby enabling diversification even in common scenarios where no semantic information on the items is available. Items are clustered based on a novel notion of priority-medoids that provides a natural balance between the need to present highly ranked items vs. highly diverse ones. We demonstrate the operation of DiRec in the context of a movie recommendation system. We show the advantage of recommendation diversification and its feasibility even in the absence of semantic information.**

I. Introduction

Online shopping has grown rapidly over the past few years. Besides the convenience of shopping directly from one's home, an important advantage of e-commerce is the great variety of items that online stores offer. However, with such a large number of items, it becomes harder for vendors to determine which items are more relevant for a given user and, given the limited size of the screen, which of these possibly relevant items should be presented first.

Much research has been devoted recently to the development of *Recommender systems*[1]. These systems predict the rating (e.g., a grade on a scale of 1 to 5) that a user would assign to an unseen item, and consider items with a high predicted rating to be relevant. But, which of these highly rated items should be presented first to the user? A naive solution would be to simply sort the items by their estimated rating and present the top-k that fit onto the screen. This however may result in an over-specialized items list. For example, suppose that a user is interested in movie recommendations. Assume that only 5 movies may fit onto the screen and that the top-5 ranked movies, for this user, all happen to be Star Wars sequels. While the given user may indeed like this series, a more *diverse* (and wider) view of the highly ranked movies may be desirable. For instance one that includes a Star Wars movie, but also other movies like Star Trek or E.T., (with the access to more Star Wars movies enabled via a "more of that" zoom-in button).

The DiRec plug-in presented in this demo provides precisely such a diversification and zoom-in facility. To support this, DiRec has to address two main challenges. The first challenge is how to measure the similarity/diversity of two given items. Previous proposals are typically based on the assumption that some semantic information on items (e.g. the genre of the movie, the director, the actors) is given. In practice, however, many recommender systems do not carry such semantic information [1]. But even when they do, it is not always clear how to define item diversity based on the given semantic information [2]. For example, some movies of the same director/leading actor may indeed be similar, whereas others may not. To overcome this difficulty, DiRec takes a different approach, inspired by work on Collaborative Filtering (CF)[3]. Instead of relying on semantic information, it defines item similarity (and correspondingly diversity) based solely on ratings that previous users gave to the items. Intuitively, each item is viewed as a vector of ratings, with vector distance (measured, e.g., by cosine, L^i distance, or Pearson correlation coefficient) used as measure of similarity/diversity.

The second challenge is the need to balance, when choosing items, between two possibly conflicting objectives: presenting highest ranked items vs. choosing highly diverse ones. Previous works attempted to resolve this by assigning a weight to each objective and selecting an items set that maximizes the weighted sum. But the question is which weights to choose?[4]. Here again DiRec resolves the problem by taking an alternative novel approach that avoids the use of weights altogether. We introduce the notion of *priority-medoids*, an adaptation of the classical notion of *medoids*[5] to a context where items have priorities (ratings). *Priority-medoids* (to be defined formally in the sequel) allow for natural clustering of items and the selection of cluster representatives that balance rank and diversity. While we show that identifying the best priority-medoids is NP-hard, we present a heuristics based on *priority cover-trees* (an adaptation of the classical cover-trees [6] to our context) which is used by DiRec and provides satisfactory results along with fast response time, as demonstrated in the demo.

DiRec is designed as a plug-in that can be deployed on CF-based recommender systems, by implementing a simple API, to diversify the recommendations presented to users. Alongside each presented item, DiRec provides a "more of that" zoom-in button that allows to view similar recommended items. Here again, no semantic information is required to identify the similar items, and they are once more presented in an as diversified as possible manner. (Users can then, again, zoom-in on each of presented items, and so on). An interesting

This work was partly supported by the EU project Mancoosi, by the Israel Science Foundation and by the US-Israel Binational Science Foundation.

property of our implementation is that the priority cover-tree constructed for the initial recommendations set contains most of the information required to support such zoom-in, thus only very minimal further computational effort is required.

Outline of the demonstration: We demonstrate the operation of DiRec in the context of a movie recommendations system. We use real data provided by Netflix [7]. As mentioned above, DiRec is deployed on a CF-based recommender system [8]. Given a user request, the underlying CF system computes the top-100 most relevant movies and supplies them, along with their ranking and similarity measures to DiRec . DiRec then determines how (and which of) the movies should be presented to the user.

To demonstrate the advantages of the diverse movies set presented by DiRec , we first compare it to the naive, sorted-by-rank movies list, that would have been provided by the CF system, had DiRec not been used. We show this list on another screen and let the audience experience navigation in the two presentations. In particular we show how, with DiRec , movie sequels and TV program episodes are naturally grouped together, represented by a single item on the screen, and can be easily reached, if desired, via the zoom-in mechanism. This is particularly interesting given the fact that no semantic information (e.g. movie names, actors, etc.) is given to DiRec . The quality of the item sets selected by algorithms will be evaluated both based on the audience input, as well as by empirical measures. Throughout the demonstration we will follow what happens under the hood, using a surveillance panel. The panel will present the constructed priority cover-tree, the resulting approximated priority-medoids and the different steps of our algorithm for choosing the item sets.

II. Technical Background

We start with some background on CF (Section II-A). Then we introduce the notion of priority-medoids (Section II-B), followed by a brief description of the priority cover-tree (Section II-C) used for their approximation. Finally we present our zoom-in mechanism (Section II-D).

A. Collaborative Filtering

CF is the process of predicting users rating to items based on previous ratings of (similar) items by (similar) users. Unlike sematic-based methods [1], CF does not use semantic properties of the items (e.g. actors, director, etc.). Instead, it is based on the assumption that users who agreed in the past on item ratings are likely to agree again in the future. A key ingredient in CF algorithms is thus the estimation of similarity between two items. Intuitively, each item is viewed as a vector of ratings in a multi-dimensional domain, where each dimension corresponds to a given user, (recording her rating of that item). Given two items, the distance between their corresponding vectors, measured, e.g., by cosine, L^i distance, or Pearson's correlation coefficient [9], is used to define the similarity between the items. (Most major systems use Pearson's Correlation Coefficient). Ratings prediction thus consists of two main steps: (1) choosing the right neighborhood for the given item (that is, the items that are considered similar to it), and (2) calculating the predicted rating by aggregating (e.g. averaging) available ratings of neighborhood items. In the reminder of this paper we use $rate(i)$ do denote the rating prediction for an item i and $dist(i, j)$ to denote the distance between items i and j. W.l.o.g. we assume below that distance values are in the range of $[0, 1]$. (When this is not the case one may naturally map the values to this range). The smaller the distance is, the more similar (and less diverse) are the items.

B. Priority-Medoid

As mentioned in the Introduction, our solution is based on the notion of *priority-medoids*, an adaptation of the classical notion of medoids to this context. To explain this, let us first briefly (and informally) recall what standard medoids are. Consider a set I of items split into k disjoint subsets, referred to as clusters. The *medoid* of a given cluster (also called the cluster's representative) is an element in the cluster s.t. the sum of the distances from it to the other items in the cluster is minimal. Other variants that consider e.g. the average, min or max distance, also exist [5]. This sum is called the cluster's weight. The classical goal is to find a clustering that minimize the overall sum of cluster weights. Note that, given a set $I_k \subseteq I$ of k items in I, the minimal-weight clustering for which the I_k items serve as representatives (medoids), is one where each item $i \in I$ is associated (clustered) with the element in I_k that is closest to it. Thus to find the best clustering one essentially needs to identify the best I_k set.

In our context we are interested in representatives with high rating. Priority-medoids therefore add the requirement that the representatives are the ones having highest rating in their corresponding clusters. More formally, consider a subset $I_k \subseteq I$ of size k of items, s.t. I_k contains, among others, an item having the highest rating in I. We will explain below why having such an item in I_k is important. For an element $i \in I$, we denote by $rep(i)$ the item within I_k satisfying the following constraints:

- the rating of $rep(i)$ is greater or equal to that of i and
- among all items in I_k satisfying the above, $rep(i)$ is the closest to i, namely there is no other $j \in I$ with $dist(i, j) < dist(i, rep(i))$.

The items with the same representative $rep(i)$ form a cluster, and thus I_k yields a clustering formation for the items of I. Note that the fact that the highest rated element in I is a member of I_k guarantees that all elements in I indeed have a cluster to which they may belong.

The quality of the obtained clustering (and thereby the quality of the set I_k that yielded the clustering), is measured, as before, by the distance of the items to the corresponding cluster representatives, namely by $\Sigma_{i \in I} dist(i, rep(i))$. When I is known from the context, we overload notation and denote this sum by $weight(I_k)$. As for standard medoids, the lower the weight, the better the clustering (and the set I_k of the representatives) is. We are thus interested in a set I_k with minimal $weight(I_k)$. However, we point out that there may be several sets with the same minimal weight, in which case

we break the tie by choosing the one where rating values are lexicographically higher.

For example, assume $k = 3$ and we have two sets $I_3 = \{i_1, i_2, i_3\}$, $I'_3 = \{i'_1, i'_2, i'_3\}$, where $weight(I_3) = weight(I'_3)$. If the rating values of the three elements in I_3 (resp. I'_3), sorted in decreasing order are $5, 4, 1$, (resp. $5, 3, 2$) then we choose I_3 over I'_3. (An alternative could be to prefer, e.g., item sets with higher average/sum of rating). Ties may still occur when distinct items have the same rating, in which case we break it arbitrarily.

Identifying the best priority-medoids is NP-hard (the proof is by reduction to the problem of finding best regular medoids, also known to be NP-hard). We thus use a heuristic based on *priority cover-trees* (an adaptation of the classical cover-trees [6] to our context) which provides satisfactory results along with extremely good performance.

C. Priority Cover-Tree

A cover-tree is a data structure originally designed to speed up of a nearest neighbor search [6]. The use of cover-trees as a tool for selecting (classical) medoids was recently proposed in [10], in the context of diversification of query results. A key difference from the present work is that item ratings were not taken into consideration. As it turns out, however, a simple modification to the algorithm of [10] allows to account for such ratings, as follows.

A *(priority) cover-tree* can be thought of as a hierarchy of levels, where each node corresponds to a specific item, and each level is a "cover" for the level beneath it. Each node in the tree is associated with an item in I. An item can be associated with multiple nodes but can appear at most once in every level l. A conventional cover-tree obeys, for all levels, the first three invariants below. A *priority* cover-tree further obeys the forth invariant.

1) (Nesting) If a node is associated with an item i, then one of its children must also be associated with i.
2) (Separation) All nodes at level l are at least $\frac{1}{2^l}$ far from one another.
3) (Covering) Each node at level l is within distance $\frac{1}{2^l}$ to its children in level $l + 1$.
4) (Priority) Each node has a rating higher or equal to that of any of its children.

The construction algorithm for cover-trees inserts nodes into the tree in an arbitrary order (see [6] for details). The construction of priority cover-trees follows exactly the same algorithm except that items are inserted *in an order that follows their rating*, from highest rated items to lowest. Interestingly, this suffices to guarantee invariant 4 above. Once the priority cover-tree is constructed, essentially the same algorithms as in [10] can be applied to the tree to identify representative items. As the algorithm selects items that are as high in the tree as possible, invariant 4 assures that the selected items have relatively high ratings. (We omit details for space constraints).

D. Recommendation Refinement

Alongside each item (representative) that is presented on the screen, DiRec offers a "more of that" zoom-in button

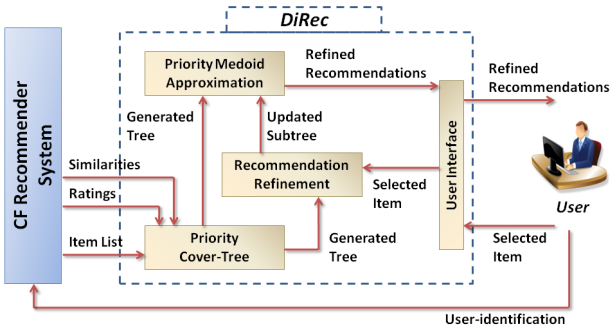

Fig. 1. DiRec architecture

that allows to view further similar items, again, presented in a diversified manner.

Here too, items similarity is determined via the distance measure mentioned above (Section II-A). Given a selected item i, we consider all items whose distance to i is smaller than their distance to the other $k - 1$ items (representatives) currently presented on the screen. Let $sim(i)$ denote this set. A straightforward approach to choose k representatives for $sim(i)$ is to apply the priority-medoids selection algorithm described in the previous subsection. To speed up the computation, DiRec uses here an optimization based on the observation that, in the previously constructed priority cover-tree, most of the elements in $sim(i)$ already appear in subtree rooted at i. We thus use this subtree as a basis for the construction of the priority cover-tree of $sim(i)$. We prune redundant elements and insert the missing ones (sorted by rating, from high to low). The insertion follows the usual procedure except that special attention is payed to cases where an element with high rating is to be inserted below one with lower rating. In such cases we first prune out the "problematic" subtree, adding its items to the "to be inserted" items priority queue. (Details omitted). While the worse case complexity of the algorithm equals to that of the naive, full priority cover-tree construction, in practice many of the items indeed appear in the subtree and only few conflicts are encountered.

III. System Overview

DiRec is implemented in Java and PHP, and designed to be deployed alongside any existing CF-based recommender system. Figure 1 illustrates the system architecture, divided into operating modules. When a user logs in, her id is passed to the *CF Recommender System* module which generates a customized list of recommended items, along with their ratings, and computes pair-wise item similarities. This information is then passed to the *Priority Cover-Tree* module, which constructs the corresponding tree. The *Priority-Medoid Approximation* module then receives this tree and uses it to select the representative items. These are presented to the user via an intuitive *User Interface* (UI). Figure 2 presents the main screen of DiRec , which shows top-5 item recommendations within the movie domain. If the user clicks on the "more of that" button, the corresponding item is passed to the *Recommendation Refinement* module. This module uses the previously generated priority cover-tree, to efficiently compute a refined tree. The tree is again passed to the Priority Medoid Approximation module for choosing item representatives.

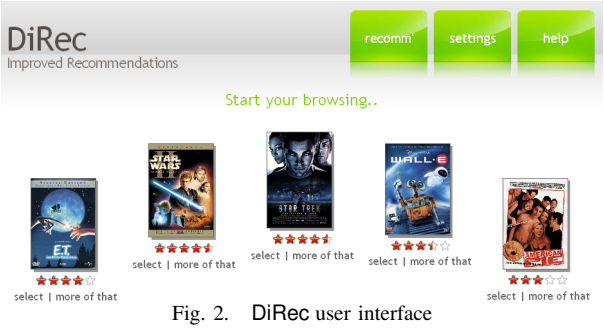

Fig. 2. DiRec user interface

We finally note that CF systems that wish to keep their existing UI may still leverage DiRec by employing it in a "behind the scenes" mode. In this case DiRec 's UI is disabled and DiRec acts as a service which is invoked by system calls.

IV. Demonstration

We start with a brief description of the system settings used in the demonstration, then describe the demonstration scenario.

System Settings: DiRec uses a CF-based recommender system [8] that we implemented. The distance measure used in our implementation for estimating items similarity is Pearson's correlation coefficient. We use in the demonstration a real data set from the cinema domain, provided by Netflix [7]. Given a user request, the underlying CF system computes the top-100 most relevant movies for the given user, and supplies them, along with their ranking and pair-wise item similarities to DiRec . The data set contains over 100 million distinct movie ratings, by approximately 500,000 users, to 18,000 different movies. This data set provides only raw user ratings (such as 1 to 5 "stars" given by individual users) and does not hold any semantic properties besides the movie names. While DiRec uses no semantic information, to evaluate the quality of our results we used the movie titles as well as information obtained from IMDB [11], to identify movie sequels and multiple episodes of the same TV program. This allows us to empirically measure whether the recommendations presented to the users include such sequels/episodes, and how many of these sequels/episodes are retrieved in the zoom-in process.

Demonstration Scenario: We demonstrate the operation of DiRec by showing the diversified recommendations presented to various Netflix users. We recall that CF-based recommender systems compute, for each user, a *personalized* set of relevant items (movies in our context). To allow the audience to better relate to, and judge, the presented recommendations, we choose Netflix users with taste similar to audience members (by asking the audience for their top favorite movies and selecting Netflix users with similar tastes).

To demonstrate the advantages of the diverse movies set presented by DiRec , we compare it to the naive, sorted by rank movies list, that would have been provided by the CF system, had DiRec not been used. (We show the latter on another screen.) We let the audience examine the differences between the recommendations presented on screen in the two settings. We then present the zoom-in mechanism by clicking on one of the item's "more of that" button. In particular, we show how, with DiRec , movie sequels and TV program

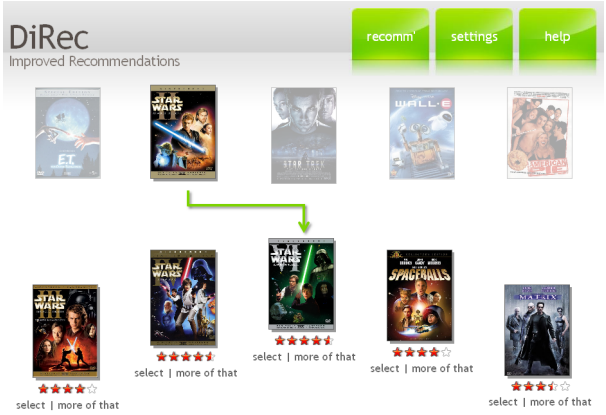

Fig. 3. DiRec "more of that" (zoom-in) mechanism

episodes are naturally grouped together. Figure 3 presents an example for such a refined recommendations set. In this specific case, continuing with the example in Figure 2, the user clicked on the "more of that" button of the "Star Wars II: Attack of the Clones" movie. DiRec successfully identifies the Star Wars sequels and presents in response three additional sequels. It also presents two additional movies that do not belong to this series, yet are related, and were chosen by DiRec to provide a more diverse set of recommendations. This is particulary interesting given that no semantic information (e.g. movie names, genres, etc.) is made available for DiRec .

The quality of the movie sets presented by DiRec and the refined sets generated by the zoom-in mechanism, will be evaluated by the audience as well as by empirical measures as described above. As mentioned in the Introduction, a surveillance panel will show what happens under the hood: the different steps of our algorithm, the constructed priority cover-trees and the resulting approximated priority-medoids.

To conclude, the DiRec plug-in demonstrated here allows CF recommender systems to diversify the recommendations that they present to users, balancing the rating and the diversity of the recommended items. DiRec provides a solution in common scenarios where semantic information is unavailable. Combining our ratings-based (quantitative) approach with a semantic (qualitative) one, when such semantic information is available, is an intriguing future research direction.

References

[1] G. Adomavicius and A. Tuzhilin, "Towards the next generation of recommender systems," *IEEE TKDE*, 2005.
[2] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen, "Improving recommendation lists through topic diversification," *WWW*, 2005.
[3] X. Su and T. Khoshgoftaar, "A survey of collaborative filtering techniques," *Advances in Artificial Intelligence*, 2009.
[4] C. Yu, L. Lakshmanan, and S. Amer-Yahia, "It takes variety to make a world: Diversification in recommender systems," *EDBT*, 2009.
[5] L. Kaufman and P. Rousseeuw, "Finding groups in data: An introduction to cluster analysis," *Wiley's Series in Probability and Statistics*, 1990.
[6] A. Beygelzimer, S. Kakade, and J. Langford, "Cover trees for nearest neighbor," *ICML*, 2006.
[7] J. Bennet and S. Lanning, "The netflix prize," *KDD Cup*, 2007.
[8] R. Boim, H. Kaplan, T. Milo, and R. Rubinfeld, "Improved recommendations via (more) collaboration," *WebDB*, 2010.
[9] J. L. Rodgers and W. A. Nicewander, "Thirteen ways to look at the correlation coefficient," *The American Statistician*, 1988.
[10] B. Liu and H. Jagadish, "Using trees to depict a forest," *VLDB*, 2009.
[11] "Imdb interface," *http://www.imdb.com/interfaces/*.