Diversification and Refinement in Collaborative Filtering Recommender [*]

Rubi Boim Tova Milo Slava Novgorodov
School of Computer Science
Tel-Aviv University
{boim,milo,slavanov}@post.tau.ac.il

ABSTRACT

This paper considers a popular class of recommender systems that are based on Collaborative Filtering (CF) and proposes a novel technique for diversifying the recommendations that they give to users. Items are clustered based on a unique notion of *priority-medoids* that provides a natural balance between the need to present highly ranked items vs. highly diverse ones. Our solution 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. It also provides a natural zoom-in mechanism to focus on items (clusters) of interest and recommending diversified similar items. We present DiRec , a plug-in that implements the above concepts and allows CF Recommender systems to diversify their recommendations. We illustrate the operation of DiRec in the context of a movie recommendation system and present a thorough experimental study that demonstrates the effectiveness of our recommendation diversification technique and its superiority over previous solutions.

Categories and Subject Descriptors: H.2.8 [Database Applications]: Data mining

General Terms: Algorithms, Experimentation

Keywords: diversification, refinement, recommendations, CF

1. 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*[2]. 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 sequels enabled via a "more of that" zoom-in button.

This papers aims to provide precisely such diversification and zoom-in facilities. Specifically, we focus here on a popular class of recommender systems that is based of Collaborative Filtering (CF), in which user ratings to items based on previous ratings of (similar) items by (similar) users[15]. A first question that needs to be addressed when designing such a diversification mechanism is how to measure the similarity/diversity of two given items. Previous proposals are often based on the assumption that some semantic information on items (e.g. the genre of the movie, the director, the actors) is given. CF recommender systems, however, typically *do not carry such semantic information* [2]. But even if they had, a problem is that it is not always clear how to define item diversity based on a given semantic information [20]. For example, some movies of the same director/leading actor may indeed be similar, whereas others may not. To overcome this difficulty, we follow the CF approach [15, 17] and instead of relying on semantic information, determine items similarity (and correspondingly diversity) based solely on ratings that previous users gave to the items. Intuitively, each item here 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.

A second important challenge is the need to balance, when choosing items, between two possibly conflicting objectives: presenting highest ranked items vs. choosing highly diverse ones. Some previous works attempted to resolve this by assigning a weight to each objective and selecting an items set that maximizes the weighted sum[9]; others used thresholds to bound the allowed similarity between items and the drop in rank [16]. But the difficult question always is *which weights or thresholds to choose*?. Indeed, a manual tuning of weights/thresholds (e.g. by experimentation) for a given data set is not only time consuming but is also no longer effective when the data changes [9]. To solve this problem we propose here a novel approach that *avoids the use of weights/thresholds altogether*. We introduce the notion of *priority-medoids*, an adaptation of the classical notion of *medoids*[11] to a context where items have pri-

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

orities (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. The clustering further allows the realization of an intuitive "zoom-in" mechanism, where users can focus on specific items on the screen and view similar recommended items. Priority-medoids sub-clustering is then be used, recursively, to diversify their presentation (and to allow further zooming-in).

To best of our knowledge, the only other previous algorithm without weights/threshold is Algorithm Greedy of [16], which does not support zoom-in. The tradeoff between ranking and diversity is hard-coded in the algorithm without any declarative notion of optimality. An advantage of a declarative definition is that it is not tied to a particular algorithm and thus allows for formal analysis and optimization. While we show that identifying the optimal priority-medoids is NP-hard, we present an efficient (ptime) heuristic based on *priority cover-trees*, a particular sub-class of cover-trees [4] that proves to be extremely effective in this context. Our experiments show that the representatives chosen by our algorithms, with no need for weights tuning whatsoever, are as good and sometimes even superior to those obtained even with optimally-tuned weights of previous algorithms. We further present an optimization technique that exploits the properties of our algorithm for an efficient realization of the above mentioned "zoom-in" mechanism.

DiRec . To demonstrate the effectiveness of our approach, we implemented the above solution in the DiRec prototype system. DiRec is designed as a plug-in that can be deployed on CF-based recommender systems, and was demonstrated in [7]. [7] provides a high-level description of the system, while the current paper presents the underlying model and algorithms.

Related Work. As explained above, a recommendations list that consists of the items with top-k predicted ratings may suffer from over specialization. Indeed, [19] evaluated the diversity of top-k items generated by traditional CF algorithms and showed it to be fairly low. Several algorithms that attempt to diversify the recommendation were proposed in the literature (see [9] for a survey). They fall generally into two classes: *greedy heuristics*, where the recommendation list is constructed "one-by-one" by maximizing a given distance function at each step (e.g. [10, 20, 16]), and *interchange (Swap) heuristics*, where an initial list is first constructed and then refined by a series of actions (e.g. [17, 16]). But common to most is the use of *predefined weights or thresholds* to determine the balances between ranking and diversity or to bound the allowed similarity between items and the drop in rank [10, 20, 17, 18]. While the selected weights/thresholds clearly affects the performance of the algorithms, their precise choice is left open in all the works we are aware of. Such use of weights/thresholds is problematic since their manual tuning (e.g. by experimentation) for a given data set is not only time consuming but is also no longer effective when the data changes [9]. As explained above, our solution employs, instead, priority medoids, to declaratively capture the desired balance. This is in contrast to [16] where the tradeoff is hard-coded in the algorithm. It further has the advantage of allowing for a natural (and optimizable) zoom-in mechanism.

The importance of results diversification has been recognized also in the context of database queries [8, 14, 12]. For example, [8] proposes a notion of diversity over structured query results which are post-processed and organized in a decision tree to help users navigate them; [14] uses attributes content to group tuples in a meaningful way that allows for convenient data exploration. This line of works however relays heavily on the *structured data content*. But such structured (semantic) information is *not available* in CF Recommender systems. Our work alleviates this problem by adopting the CF approach and relying on CF (dis)similarity measures rather than semantic ones.

The most relevant to our work, although also targeted to structured databases, is [12], where the authors used the notion of classical medoids (approximated by classical cover-trees) to select representatives for the query results and to zoom in on similar answers. While our work was inspired by [12], a key difference is that [12] *completely ignores tuples rating/priority*. We will see that our use of *priority* medoids and, resp., *priority* cover-trees, over the classical ones, prove to be extremely effective and greatly improves that generated recommendations.

Contributions. The technical contributions of this paper can be summarized as follows:

- We introduce the novel notion of priority-medoids as a tool for selecting item representatives. Our approach naturally balances the rating and the diversity of the recommended items and is applicable even in the absence of semantic information (as often is the case is CF recommender systems).
- We show that finding optimal priority-medoids is NP-hard and provide an alternative effective heuristic based on priority cover-trees.
- We exploit the properties of our algorithm to design an efficient, incremental zoom-in mechanism that allows to focus on individual items, identify their neighborhood (similar) items and select appropriate representatives for them.
- We demonstrates the superiority of our solution relative to previous algorithms as well as its efficiency.

The paper is organized as follows. Section 2 presents priority-medoids. Section 3 then explains how they are approximated and how item representatives are selected. Section 4 describes the zoom-in mechanism,and our experiments are described in Section 5 . Finally, we conclude in Section 6. For space constraints, some details of the algorithms and the experiments are omitted and can be found in the full version of the paper [6].

2. PRIORITY-MEDOIDS

We start by providing the needed background and notation for Collaborative Filtering (CF). We then consider the problem of balancing the ratings and the diversity of the recommended items and define priority-medoids.

2.1 Collaborative Filtering

Common CF algorithms are *item-based*, consisting of two main steps: (1) choosing for each item a *neighborhood* of similar items, and (2) predicting the rating that a user u will give to an item i using some aggregation function on the actual ratings, gave by u, to the items within the neighborhood of i [15]. Symmetric *user-based* variants also exist but are less frequently used [15]. A key ingredient in the algorithm is thus the estimation of similarity between two items. Intuitively, each item is viewed as a vector of ratings in a multi-dimensional speace, where each dimension corresponds to a given user, (recording her rating of that item). The distance between item vectors is then used as a measure for the items similarity. In principle, any distance measure can be employed here (e.g. Cosine or L^i distance) but Pearson's correlation coefficient [13] seems to be the preferred choice in most major systems. The basic intuition behind Pearson's measure is to give a high similarity score for two items that tend to be rated the same by many users. In the reminder of this paper we use $rate(u, i)$ do denote the predicted

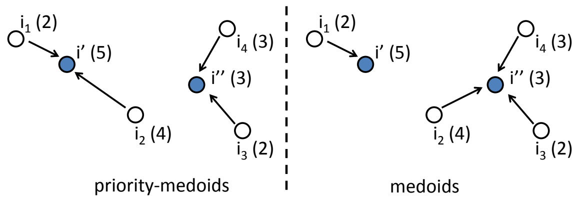

Figure 1: Priority-medoid vs. standard medoid

rating of a user u to item i. When u is known from the context we omit it and simply write $rate(i)$. We use $dist(i, j)$ 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.

2.2 Balancing Rating and Distance

We can now describe our main problem: given a set I of items and a size k, where the former holds the most relevant items for a given user (as decided by the given CF recommender system) and the latter denotes the number of items fitted onto the screen, we need to choose the subset $I_k \subseteq I$ (of size k) that will be presented to the user. A naive solution simply selects the top-k items with the highest $rate()$ values, namely the ones who maximize the sum $\sum_{i \in I_k} rate(i)$. This however may result in an over-specialized subset, as described in the Introduction. Thus, we should consider the diversity of the items as well, which may be analogically measured by the sum $\sum_{i,j \in I_k} dist(i, j)$. As these are two opposing measures, one needs to provide a good balance between the two. Additionally, this subset should also provide a good coverage for the entire set I, which intuitively can be viewed as a classic clustering problem, where we need to minimize the distances between the items in I and the "center" of the cluster they belong to.

Our solution, as mentioned in the Introduction, 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 [11]. 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. Thus, when considering a set I_k of priority-medoids, the clusters it forms are different than the ones formed when considering standard medoids.

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 two constraints:

- the rating of $rep(i)$ is greater or equal to that of i
- among all items in I_k satisfying the above, $rep(i)$ is the closest to i, namely there is no other $j \in I_k$ 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. We refer to the items in I_k as the *priority-medoids* of their corresponding clusters. 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.

EXAMPLE 2.1. *The example in Figure 1 illustrates the difference between the cluster formation in regular medoids and that of priority-medoids. Assume that $k = 2$ and that the set I_k consists of the two items i' and i''. Also assume that the euclidian distance between items describes their similarity. The number in parenthesis, next to each item, describes its rating. On the right (resp. left) hand side we see the clusters formed when the items in I_k are treated as regular (resp. priority-) medoids. The arrows outgoing the elements $i_1 - i_4$ point to their cluster representative. On the right hand side (regular medoids) items i_2,i_3 and i_4 are clustered with i'', as they are closer to it than to i'. i_1 is clustered with i'. In contrast, on the left hand side (priority-medoids), the clustering formation is different as the item ratings are now also being considered. Specifically, since the rating of i'' is lower than that of i_2, i_2 is clustered with (and represented by) i', that has higher rating, even though it is slightly further.*

The quality of the obtained clustering (and thereby the quality of the set I_k of priority-medoids 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 use a simplified 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 priority-medoids) 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.

We can show that identifying the best priority-medoids is NP-hard (proof is omitted). We thus use a heuristic, based on *priority cover-trees* - an adaptation of the classical cover-trees [4] to our context. We explain this next.

3. PRIORITY COVER-TREE

A cover-tree is a data structure originally designed to speed up a nearest neighbor search [4]. The use of cover-trees as a tool for selecting (regular) medoids was recently proposed in [12], in the context of diversification of query results. A key difference from the present work is that *item ratings were not taken into consideration*. We next show that a fairly simple modification to the algorithm of [12] allows to account for such ratings and thereby gradely improve the quality of the generated recommendations.

A conventional 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. (The root is at level zero, its children at level one, and so on). 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. In our algorithms we use a special sub-class of these trees, which we call *priority* cover-trees, that further obey 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.

An example for such a tree can be found in [6]. Next, we explain the role that invariant 4 plays in our algorithms for constructing priority-cover trees and for selecting items representatives, based on the constructed tree.

3.1 Construction

We first recall the conventional algorithm for cover-tree construction [4], then explain the needed modifications.

Given a set I of items, the standard algorithm first selects one item, arbitrarily, to serve as the root of the tree. Then it inserts the other items, iteratively, one by one (again in an arbitrary order), into the tree using an `Insert Single Item` function. For each item, the function finds the upper-most level into which the given item can be inserted, in a recursive fashion. Intuitively, it traverse the tree top-down, level-by-level, where at each step it maintains a set of nodes which can act as possible ancestors for the given item. These nodes are the ones whose descendants may potentially preserve the first three cover-tree invariants describe above. The given item is finally added once the algorithm reaches a point in the tree where all these invariants hold.

Now, to construct our *priority* cover-tree, only two simply changes to this algorithm are required:

- *Ordered Insertion.* In the conventional cover-tree construction algorithm, items are inserted in an arbitrary order. Instead, our algorithm first sorts the items w.r.t their ratings, then inserts them in descending order.
- *Tight Insertion.* In the original algorithm, at the point when the algorithm finally inserts the given item into the tree, it may have several candidate nodes that may act as the item's parent. As any node in this set is a 'legal' parent (in the sense that the invariants will be preserved), the original algorithm chooses among them arbitrary. In contrast, our refined algorithm always prefers the node with the smallest $dist()$ measure to the inserted items.

For space constraints the full details of the algorithm, as well as its correctness proof, are deferred to [6]. We only note here that ordered insertion suffices to guarantee Invariant 4. Tight insertion, on the other hand, is not mandatory for the correctness of the construction. However it will prove useful later, when considering recommendations refinement (zoom-in). We discuss this issue in more detail in Section 4.

3.2 Representative Selection

We next explain how the constructed priority cover-tree is used to select item representatives. We use below the following notations. Given a node n in the tree and an integer $l \geq 0$, we say that n is at level l in the tree, if the distance between n and the root of the tree (counted by the number of edges on the path between them) is l. We use C_l to denote the set of nodes at level l.

The pseudo-code of the algorithm is given in [6] and we sketch below the key ideas. Consider a set I of items which the CF recommender determined as most relevant for a given user. As explained above, the k items that we want to present to users are the priority-medoids of I. To find such a subset, with low weight, we use the priority cover-tree of I as follows: Starting from the tree root, we search for the first level l within the tree that includes at least k nodes, namely where $|C_{l-1}| < k$ and $|C_l| \geq k$. We then choose our k representatives from within C_l. Recall that due to the structure of the priority cover-tree the nodes in C_i are at least $\frac{1}{2^l}$ far from one another and have rating higher than their descendants. If C_l contains exactly k nodes, then we simply choose these nodes for representatives and we are done. Otherwise, we need to select a subset of size k. To select good representatives, we use the tree structure: Recall the *nesting* property of the tree, which implies that $C_{l-1} \subseteq C_l$. Moreover, because of the hierarchical structure of the tree, the nodes (items) within C_{l-1} have rating $\geq$ than those in C_l and are pairwise further apart from each other than from the remaining nodes. Thus, we first select the nodes (items) in C_{l-1}, then add the remaining $k - |C_{l-1}|$ from within $C_l - C_{l-1}$. We consider in our implementation and experiments three alternative methods for choosing these additional elements:

- *Max-rating* - the elements having the maximal ratings,
- *Max-diversity* - the elements that are farthest from the previously chosen elements, and
- *Max-coverage* - the elements having the maximal number of descendants.

Note that, during the tree construction, we can easily record the number of descendants each node has, and use this information for the implementation of the Max-coverage variant. Thus, all three options are equivalent in terms of computational effort. They may differ however, as we shall see later, in the quality of the generated representatives set.

4. RECOMMENDATION REFINEMENT

Alongside each item (representative) i that is presented on the screen, DiRec offers a "more of that" zoom-in button that allows the user to view further related items.

One possible approach for identifying such items in I is to select the ones whose distance from i (as measured by the $dist()$ function) is below a certain threshold. However, as it is never obvious which threshold should one choose, we take, instead, an alternative approach based on the following intuition: when a user clicks on a specific item i, she not only signals her interest in item's i family but also signals that i interests her more than the other $k-1$ presented representatives. Recalling the clustering formation generated by the presented representatives (priority-medoids), we determine the items in the cluster represented by i as relevant, denoting this set by $relevant(i)$.

Here again, the set $relevant(i)$ often contains more items than could fit on the screen and a subset needs to be chosen, to be presented to the user.

A straightforward approach to choose k representatives for $relevant(i)$ is to build a priority cover-tree for the set, then use it to select representatives, as described above. Rather than doing so, we employ instead an optimized, significantly more efficient, algorithm that is based on the following observation: In the priority cover-tree previously constructed for I, most of the elements in $relevant(i)$ already appear in subtree rooted at i. Indeed, the *tight Insertion* used in to the construction algorithm (Section 3.1) was employed precisely to increase the number of such elements. We thus use this subtree as a basis for the construction of the priority cover-tree of $relevant(i)$. Note however that the subtree may not include all the members of $relevant(i)$ and also it might include some redundant elements that are not members of $relevant(i)$. Full details of the algorithm is available at [6].

5. EXPERIMENTS

We have implemented the above algorithms in DiRec [7], a plug-in that allows CF Recommender systems to diversify the recommendations that they present to users. We used DiRec in a variety of settings to evaluate the described algorithms by (1) a user study, (2) empirical evaluations, (3) performance measures and (4) refinement process. Due to space constraints, we present in this paper the results only for (2) and a short summary for (1). Nevertheless, the full description of all experiments are available at [6] for the intrigue readers.

Experiments setting. In our experiments, DiRec was employed on top of a CF-based recommender system (C2F [5]). For estimating item similarity, C2F (and thus DiRec) uses Pearson's correlation coefficient [13]. We used a natural linear inverse mapping to compute items distance out of their Pearson correlation value: $dist(i, j) = (1 - Pearson(i, j))/2$. The experiments were preformed on an Intel quad-core machine (Q9400) with 2.66 GHz CPUs, (using only a single core of the CPU), 4GB memory and windows XP x64 edition. As data, we used a real-life data set from the cinema domain, provided by Netflix [3], which contains over 100 million distinct raw movie ratings (such as 1 to 5 "stars"), by almost 500,000 users, and no semantic information on the movies.

Algorithms. Our algorithm for representatives selection first constructs a priority cover tree, then chooses representatives out of this tree. We consider the three variants of this choice: Max-rating, Max-diversity and Max-coverage, denoted below *PCT-R*, *PCT-D* and *PCT-C*, respectively. (PCT stands for Priority Cover-Tree). As we have mentioned, our use of *priority* cover-tree, instead of the classical cover-tree of [12], allows to take items rating into consideration. To illustrate the resulting improvement in the quality of the generated recommendations we had also implemented the cover-tree algorithm of [12], denoted below *CT*.

Recall that previous works typically use semantic information, to diversify items, and predefined weights/thresholds, to balance between rating and diversity. Since no sematic information is available in our context, one can employ here only algorithms that operate without it. We have implemented the state of the art such algorithms from [16] (*Greedy* and *Swap*) and compared them to ours. Note that Algorithm Swap, like all other previous works (except for Algorithm Greedy), use predefined thresholds to balance rating and diversity. We had thus first optimized the thresholds ([6]) and had our algorithms compete against these optimized versions. The results for *Greedy* and *Swap* were similar and we thus show below only those of *Swap*. (Interested readers can find a brief description of the algorithm in [6]).

Finally, as a base line, we had also implemented the two "extreme case" algorithms, *optR* (for Optimal Rating), which operated like a standard CF algorithm, selecting the k items having highest rating and ignoring diversity, and *optD* (for Optimal Diversity), which ignores the rating of items and selects k items whose pairwise distance values is the highest. [1]

[1]Note that *optD* require solving an NP-hard problem [9]. Its EXP-time algorithm iterates over all subsets of size k to find the best one. In our experiments this became infeasible for items sets of size $\geq$ 100, and we thus show results for them only up to that size.

User Study. To assess the quality of the item sets that DiRec generates (and thereby the quality of our algorithms), we ran a set of experiments with 50 volunteers who were asked to evaluate the quality of the presented movie recommendations. Due to space constrains, we only note the final results of the experiment (for a full description see [6]). Algorithm *PCT-R* got the highest results, with *PCT-C*, *Swap* and *PCT-D* not so far behind. Not surprisingly, these four algorithms got much higher results than *optR*, *optD* and *CT*, as they consider both the relevance and the diversity of the items (while the others focus solely on one property). While the grade difference between *PCT-R* and *Greedy* is smaller, recall that the former has further the advantage of supporting a natural zoom-in mechanism.

5.1 Empirical Evaluation

To better understand the results and how the *PCT* algorithms (and the preferred *PCT-R* in particular) balance ranking and diversity, we run a second set of experiments over the Netflix data where we analyzed the quality the item sets generated by all the algorithms, in terms of their rating and diversity. In each set of experiments reported below we randomly chose a set of 1000 users from the Netflix data set and run the experiment for each of them. The results presented here are the average of all 1000 users. (The variance was less that 5%). We ran all experiments for varying size I of item candidates returned by the CF-system (I ranging from 10 to 1000) and varying number k of representative items selects out of them (k ranging from 3 to 20). The results were generally independent of the number k of selected representative. Thus, except when stated otherwise, we present below a representative sample for $k = 5$, the common number of recommendations given by typical recommender systems. For a generated set I_k of representatives, we evaluated its quality w.r.t to the following measures.

Rating. We measure the relevance of the suggested set of item I_k to the current user by the sum of the element ratings, i.e. $rating(I_k) = \sum_{i \in I_k} rate(i)$. (Higher value means better results).

Diversity. In the absence of semantic information, the diversity of the items in I_k may be measured by the pairwise-distances of the items, namely $diversity(I_k) = \sum_{i,j \in I_k} dist(i, j)$. (Here too, higher value means better results).

Although our algorithms do not require semantic information, it is interesting to examine the actual semantic diversity of the selected items. To get a sense of this, we extracted for each movie information from the Internet Movie Databases IMDb [1] and used it to determine, for each movie, to which of series it belongs (if any). We also extracted the movie's genre (action, drama, etc.). For diversification, it is intuitively preferable not to have multiple movie sequels in the set presented to the user, and to have movies from different genres. We quantify this as follows.

- We denote below by $sequel(i)$ the series to which an item (movie) i belongs. We count the number of distinct series to which the movies in I_k belong and divide this by k, the number of movies in I_k. Namely, $sequelDivers(I_k) = \frac{|\{sequel(i)|i \in I_k\}|}{k}$. Observe that $sequelDivers$ is a number in the range of $(0 : 1]$. Higher values reflect higher diversity, hence better result.

- We assume some total order (e.g. lexicographical) on the possible move genres and denote by $genres(i)$ a boolean vector that records the (possibly multiple) genres of the movie i. (A value 1 in entry j means that the movie i belongs to genre i). To measure diversity we consider the pairwise dis-

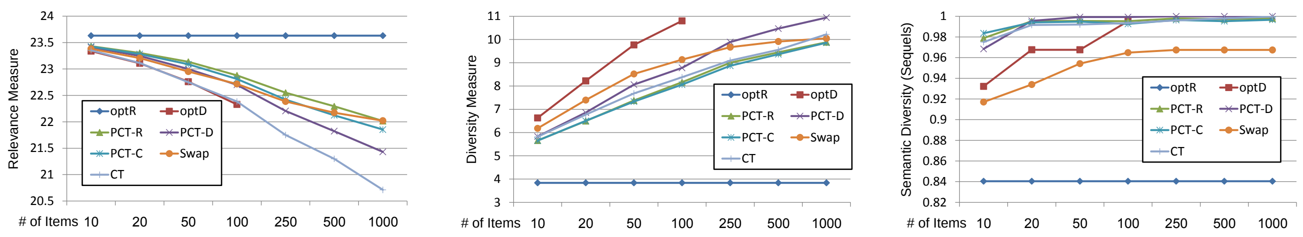

Figure 2: (a) Rating measure (b) Distance-based diversity (b) Semantic (sequel) diversity

	Rating	Distance-based diversity	Sequel diversity
PCT-D	-	+	=
PCT-C	-	=	=
Swap	-	+	-
CT	- -	=	=
optR	++	- -	- -
optD	- -	++	-

Table 1: Summary of the results (relative PCT-C)

tances (by the Cosine measure) between the vectors. Then, $genreDivers(I_k) = 1 - average\{cosine(genres(i), genres(j)) \mid i, j \in I_k\}$. Here again $genreDivers$ is a number in the range of $(0:1]$ with higher values reflecting higher diversity.

Figure 2 shows the values for the relevance, distance-based diversity, and sequel-based diversity, for the various algorithms, for varying sizes of item sets I. The results for genre-based diversity are similar to those of sequel-based one and are thus omitted for space constraints. The figure depicts these values, for the sets computed by the seven algorithms mentioned before: *PCT-R*, *PCT-D*, *PCD-C*, *Swap* (with its tuned threshold), *CT*, *optR* and *optD*. k here equals 5 and the size of the set I ranges from 10 to 1000.[2]

Recall that higher values here mean better results. Diversity generally increases when I grows, as there are more items to choose from; Rating decreases, as more diverse but lower ranked items are chosen. Table 1 provides a summary of the graphs in Figure 2 that highlights how the various algorithms perform (for the various measures) relative to *PCT-R*, and helps to explain why users may have found its results superior. For an algorithm A and measure M, an "+" (resp. "++") entry indicates that Algorithm A generally performed slightly (significantly) better than *PCT-R* in measure M. Similarly, "$-$" (resp. "$--$") indicates that A performed slightly (significantly) worse than *PCT-R* in M. An "=" entry indicates similar performance.

As expected *optR* have the highest rating value and lowest diversity. Analogously, *optD* has lowest rating value and highest distance-based diversity. *CT* too has lowest rating value as it ignores item ranks. An interesting point to observe is that although some algorithms (and in particular *optD*) achieve better distance-based diversity than *PCT-R*, no algorithm achieves better sequel diversity. This discrepancy occurs, intuitively, because *optD* tries to maximize the *total sum* of distances without considering the (minimal) distance between two individual representatives (and thus may select two sequels in the same set). For a concrete example see [6]. The same phenomenon occurs more generally in many real life cases and explains why *PCT-R* is not handicapped by its non optimal distance diversity. Indeed, its sequel diversity is close to 1 (meaning that, as desired, the result contains no sequels) and we can see that no other algorithm was able to achieve better semantic (sequel) diversity. Furthermore, among the algorithms that achieve sequel diversity equal to that of *PCT-R*, it is the one with highest rating score, which can explain why its recommendations were more favorable than the rest.

[2]Recall that *optD* became infeasible for items sets of size greater than 100, and we thus show results for them only up to that size.

6. CONCLUSION

The DiRec plug-in presented in this paper allows CF recommender systems to diversify the recommendations that they present to users. It is based on a novel notion of priority-medoids that declaratively balances the rating and the diversity of the recommended items. We introduced priority cover-trees as a tool for efficient selection of item representatives. Our solution further allows for an effective realization of a natural "zoom-in" mechanism, presenting to users similar yet diversified items. Our experiments demonstrated the effectiveness of our recommendation diversification technique and its superiority over previous proposals.

DiRec provides an effective solution in common scenarios where semantic information is unavailable. Combining our ratings-based (quantitative) approach with a semantic (qualitative) one, when such semantic data is available, is an intriguing future research direction.

7. REFERENCES

[1] Imdb interface. *http://www.imdb.com/interfaces/*.

[2] G. Adomavicius and A. Tuzhilin. Towards the next generation of recommender systems. *IEEE TKDE*, 2005.

[3] J. Bennet and S. Lanning. The netflix prize. *KDD Cup*, 2007.

[4] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. *ICML*, 2006.

[5] R. Boim, H. Kaplan, T. Milo, and R. Rubinfeld. Improved recommendations via (more) collaboration. *WebDB*, 2010.

[6] R. Boim, T. Milo, and S. Novgorodov. Diversification and refinement in collaborative filtering recommender (full version). *http://www.cs.tau.ac.il/boim/publications/cikm11-full.pdf*.

[7] R. Boim, T. Milo, and S. Novgorodov. Direc: Diversified recommendations for semantic-less collaborative filtering. *ICDE*, 2011.

[8] Z. Chen and T. Li. Addressing diverse user preferences in sql-query-result navigation. *SIGMOD*, 2007.

[9] M. Drosou and E. Pitoura. Search result diversification. *SIGMOD Record*, 2010.

[10] S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. *WWW*, 2009.

[11] L. Kaufman and P. Rousseeuw. Finding groups in data: An introduction to cluster analysis. *Wiley's Series in Probability and Statistics*, 1990.

[12] B. Liu and H. Jagadish. Using trees to depict a forest. *VLDB*, 2009.

[13] J. L. Rodgers and W. A. Nicewander. Thirteen ways to look at the correlation coefficient. *The American Statistician*, 1988.

[14] J. Stoyanovich, S. Amer-Yahia, and T. Milo. Making interval-based clustering rank-aware. *to appear in EDBT*, 2011.

[15] X. Su and T. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 2009.

[16] C. Yu, L. Lakshmanan, and S. Amer-Yahia. It takes variety to make a world: Diversification in recommender systems. *EDBT*, 2009.

[17] C. Yu, L. Lakshmanan, and S. Amer-Yahia. Recommendation diversification using explanations. *ICDE*, 2009.

[18] M. Zhang and N. Hurley. Avoiding monotony: Improving the diversity of recommendation lists. *RecSys*, 2008.

[19] M. Zhang and N. Hurley. Evaluating the diversity of top-n recommendations. *ICTAI*, 2009.

[20] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. *WWW*, 2005.