Noname manuscript No.
(will be inserted by the editor)

A Supervised Active Learning Framework for Recommender Systems based on Decision Trees

Rasoul Karimi · Alexandros Nanopoulos · Lars Schmidt-Thieme

Received: date / Accepted: date

Abstract A key challenge in recommender systems is how to profile new users. A well-known solution for this problem is to ask new users to rate a few items to reveal their preferences and to use active learning to find optimally informative items. Compared to the application of active learning in classification (regression), active learning in recommender systems presents several differences: although there are no ratings for new users, there is an abundance of available ratings –collectively– from past (existing) users. In this paper, we propose an innovative approach for active learning in recommender systems, which aims at taking advantage of this additional information. The main idea is to consider existing users as (hypothetical) new users and solve an active learning problem for each of them. In the end, we aggregate all solved problems in order to learn how to solve the active learning problem for a real new user. As the ratings of existing users (i.e., labels) are known and are used for active learning purposes, the proposed framework is in fact a supervised active learning framework. Based on this framework, we investigate two different types of models: the first model is based on information about average item ratings and the second on matrix factorization. We present experimental results on the Netflix dataset, which show that the proposed approach significantly outperforms state-of-the-art baselines.

Keywords Active Learning · Recommender Systems · Cold-Start Problem · Matrix Factorization

Rasoul Karimi, Lars Schmidt-Thieme
Information Systems and Machine Learning Lab Marienburger Platz 22 University of Hildesheim 31141 Hildesheim Germany
Tel.: +49 5121 / 883-40360
Fax: +49 5121 / 883-40361
E-mail: karimi, schmidt-thieme@ismll.uni-hildesheim.de

Alexandros Nanopoulos
Department of Business Informatics., Schanz 49, University of Eichstatt-Ingolstadt , 85049 Ingolstadt, Germany
Tel.: +49 841937-1872
Fax: 0049 841937-1870

1 Introduction

Recommender systems help web users to address information overload in a large space of possible options (Adomavicius and Tuzhilin, 2005). In many applications, such as in e-commerce, users have too many choices and too little time to explore them all. Moreover, the exploding availability of information makes this problem even tougher.

There are several techniques for generating recommendations. Collaborative filtering is a traditional technique that is widely applied (Burke, 2002; Konstan et al, 1997; Rashid et al, 2002). It makes automatic predictions about the interests of a user, by exploiting information about preferences of other users. The underlying assumption of the collaborative filtering approach is that users with similar preferences in the past will tend to have similar preferences also in the future. Collaborative filtering methods fall into two categories: memory-based algorithms and model-based algorithms. In memory-based techniques, the value of the unknown rating is computed as an aggregate function of the ratings of some other (usually, the top-N most similar) users for the same item (Konstan et al, 1997). Model-based collaborative techniques provide recommendations by estimating parameters of statistical models for user ratings. Nevertheless, recent research, especially as has been demonstrated during the Netflix challenge[1], indicates that Matrix Factorization (MF) is a superior prediction model compared to other approaches (Koren et al, 2009). MF maps users and items into a latent space and then items that are in the neighborhood of the target user in the latent space are recommended to her.

Despite its performance, collaborative filtering suffers from the *cold-start* problem, which can be categorized as new-user, new-item, or in general new-system cold-start problem.[2] Although both the new-user and the new-item problems are equally important, in this paper we focus on the new-user problem, since it affects every new user that enters the system and, thus, it can directly impact the success of the system by improving the experience of the new users at this critical phase, which can reduce the chances of churn. A simple and effective way to overcome the new-user problem is by posing queries to new users, in order that they express their preferences about selected items, e.g., by rating them. Nevertheless, the selection of items must take into consideration that users are not willing to answer a lot of such queries. Thus, new users should be asked to give ratings just to a few items that are *informative*, i.e., will help the most in determining their interests.

The corresponding problem in supervised learning arises, when not enough labeled data exist to train a classification (regression) model. However, when a set of unlabeled (pool) data is available, it is possible to ask a so-called 'oracle' to reveal their labels. Since querying the labels can be costly, only a few such queries can be selected. Techniques which deal with this problem are called *active learning* methods (Cohn et al, 1996), which try to select the best query as the one which is most informative, i.e., the query which reduces the test error as much as possible. There are many supervised learning (classification) problems in which active learning can be used, e.g., text classification (Yang et al, 2009; Tong and

[1] www.netflixprize.com

[2] The new-system problem refers to a recommender system that is in the early stage and has not been used by many users. In such a case, any collaborative filtering approach becomes ineffective.

Koller, 2002), where, for instance, news articles can be classified to categories such as sports, economy, science, etc. Nevertheless, most applications today involve huge amounts of data and obtaining labels, e.g., by human experts, for the complete set of data is practically not possible. Therefore, active learning provides a solution to this problem, by drastically reducing the amount of required labeled data without compromising the classification accuracy.

Compared to the application of active learning in classification (regression), active learning in recommender systems presents a crucial difference: although there are no ratings for new users, there is an abundance of available ratings –collectively– from existing (training) users. Our approach thus aims at taking advantage of this additional source of information. This is attained by considering past users as (hypothetical) new users, in order to learn the right queries to be asked to new users for active learning purposes. On one hand, as the ratings, which can be considered equivalent to labels in classification (regression), of existing users are known, we are dealing with a supervised problem. On the other hand, those ratings are used to find informative queries to new users. For these reasons, we consider the proposed framework as a supervised active learning framework.

After introducing the general framework of our approach, which opts to find only one query, we extend it based on decision trees to find a sequence of queries. Decision trees build classification (regression) models in the form of tree structures. They are built top-down from a root node by partitioning the data into subsets that contain instances with similar target values (homogeneous). Usually entropy is used to calculate homogeneity. The learning algorithm searches the space of possible branches to find the split that gives the most homogeneous branches (i.e., with the lowest entropy). In this paper, decision trees are used to develop adaptive active learning methods, i.e., new users receive different queries according to their responses to the previous queries.

The rest of this paper is organized as following: in Section 2 the related work is reviewed. Problem settings and MF are explained in Section 3. The proposed approach is introduced in Section 4, followed by the experimental results in Section 5. Finally, we conclude the paper in Section 6.

2 Related Work

2.1 Methods based on Collaborative Filtering

Active learning, in the context of the new-user problem, was introduced by (Kohrs and Mérialdo, 2001). They proposed variance and entropy for nearest-neighbor collaborative filtering. Although they do not explicitly call their approach active learning, their goal is to learn the new users' preferences with minimum questions, which is equivalent to what active learning does in classification problem. (Rashid et al, 2002) expanded this work, by considering the popularity of items and also personalizing the item selection for each individual user. Personalized item selection takes into account the fact that users may choose not to respond to all queries and, thus, there is a need to select items for which users will respond.

(Boutilier et al, 2003) was the first work that studied the new-user problem from active learning perspective. They proposed a method for selecting informative items that lead to the largest change in the estimation of ratings. (Jin and

Si, 2004) developed a new active learning algorithm based on the Aspect Model (AM), which is similar to applying active learning for parameter estimation in Bayesian networks (Tong and Koller, 2001). They use the entropy of the model as loss function, but do not directly minimize the entropy loss function, because the current model may be far from the true model and relying only on the current model can become misleading. To overcome this problem, (Tong and Koller, 2001) proposes to use a Bayesian network to take into account the reliability of the current model. This Bayesian approach is, however, intractable, because it needs excessive execution time. (Harpale and Yang, 2008) extended (Jin and Si, 2004) by relaxing the assumption that a new user can provide a rating for any queried item. This approach personalizes active learning to the preferences of each new user, as it queries only those items for which users are expected to provide a rating. (Karimi et al, 2011a) applied the most-popular-item selection to AM. The results show that it competes in accuracy with the Bayesian approach while its execution time is orders of magnitude faster than the Bayesian method.

(Karimi et al, 2011b) developed a non-myopic active learning that capitalizes explicitly on the update procedure of the MF model. Initially, this method selects those items that, when the new user's features are updated with the provided rating (i.e., after querying the selected items), the user's features will be changed as much as possible. Being inspired from optimal active learning for the regression task, (Karimi et al, 2011c) exploits the characteristics of MF and develops a method which approximates the optimal solution for recommender systems. (Karimi et al, 2012) improved the most-popular item-selection according to the characteristics of MF. They proposed a method that finds similar users to the new user in the latent space and then selects the item that is most popular among similar users.

2.2 Methods based on Decision Trees

The idea of using decision trees for cold-start recommendation was proposed by (Rashid et al, 2008), whereas (Golbandi et al, 2011) improved it, by advocating a specialized version of decision trees that adapt the preference elicitation process to the new user's responses. Like (Kohrs and Mérialdo, 2001; Rashid et al, 2002), these works do not explicitly treat the new-user problem as an active learning problem. As our method relies on (Golbandi et al, 2011), we provide a brief explanation (following other authors, we call the method of (Golbandi et al, 2011) *Bootstrapping*): Each interior node is labeled with an item $i \in I$ and each edge with users' response to item i. The new-user preference elicitation corresponds to following a path starting at the root, by asking the user to rate items associated with the tree nodes along the path and traversing the edges labeled by the user's response until a leaf node is reached. Decision trees are ternary. Each internal tree node represents a single item on which the user is queried. After answering the query, the user proceeds to one of the three subtrees, according to her answer. The answer is either "Like", "Dislike", or "Unknown." The "Unknown" means users did not rate the queried item (e.g., because they did not know it). Allowing users not to rate a queried item is crucial, because this is expected in real applications. Each tree node represents a group of users and predicts item ratings by taking the average of ratings among corresponding users.

Formally, let t be a tree node and $U_t \subseteq U$ be its associated set of users. $\mathcal{D}^t$ denotes a subset of $\mathcal{D}^{\text{train}}$ which belong to node t:

$$\mathcal{D}^t := \{(u, i, r) \in U \times I \times R \mid u \in U_t\},$$

where the profile of item i in node t is denoted as $\mathcal{D}_i^t$:

$$\mathcal{D}_i^t := \{(u, r) \in U \times R \mid u \in U_t\},$$

The predicted rating of item i at node t is computed using the item average method:

$$\hat{r}_{ti} = \frac{\sum\limits_{(u,r) \in \mathcal{D}_i^t} r + \lambda_1 \hat{r}_{si}}{|\mathcal{D}_i^t| + \lambda_1} \tag{1}$$

To avoid over-fitting, the prediction for item i is regularized towards its prediction in the parent node $\hat{r}_{si}$. λ_1 is the regularization factor. The effect of the regularization for item i becomes more significant when the number of ratings in item's profile $\mathcal{D}_i^t$ is small. The squared error associated with node t and item i is: $(e_i^t)^2 = \sum\limits_{(u,r) \in D_i^t} (r - \hat{r}_{ti})^2$. Also, the overall squared error at node t is: $(e^t)^2 = \sum\limits_{i \in I} (e_i^t)^2$.

Building decision trees is done in a top-down manner. For each internal node, the best splitting item is the one which divides the users into three groups, such that the total squared prediction error is minimized. This process continues recursively with each of the subtrees and in the end all users are partitioned among subtrees.

Suppose we are at node t. Per each candidate item i, three candidate child nodes are defined: $tL(i)$, $tD(i)$, $tU(i)$ representing users who like item i, dislike it, and have not rated it, respectively. The squared error associated with this item is $Err_t(i) = (e^{tL})^2 + (e^{tD})^2 + (e^{tU})^2$. Among all candidate items, the item which minimizes the following equation is the best :

$$\text{splitter(t)} = \underset{i \in I}{\text{argmin}} \ Err_t(i) \tag{2}$$

A naive construction of the tree would be intractable if the number of items and ratings is large. Therefore, (Golbandi et al, 2011) proposes to expand "Unknown" child nodes in a different way, by using some statistics collected from "Like" and "Dislike" child nodes. Similar to (Golbandi et al, 2011), our approach is based on decision trees, but with a different optimization function. The difference between our approach and (Golbandi et al, 2011) is presented in section 4.1.

(Zhou et al, 2011) modified (Golbandi et al, 2011) by proposing functional Matrix Factorization (fMF) which associates matrix factorization to decision trees. First, item features are initialized randomly. Then, decision trees are built. Each node of the tree represents a group of users who share the same user features. After learning the tree, the item features are updated using the learned user features. The loop continues until convergence is reached.

We consider that the method of (Zhou et al, 2011) is too expensive both in terms of time and memory costs. The computational complexity for constructing decision trees is $O(N \sum_{u \in U} |D_u|^2 + l|I|k^3 + l|I|^2k^2)$ where N is the depth of the tree, $|D_u|$ is the number of ratings by user u, l is the number of nodes, $|I|$ is the number of items, and k is the number of latent features (Zhou et al, 2011). As an illustrative example, for the Netflix dataset it holds that: $\sum_{u \in U} |D_u|^2 \approx 6.48^{10}$, $|I| \approx 18$k, and assume that $k = 50$. The number of nodes l is $\sum_{i=0}^{q} 3^i$, so assuming that $q = 7$, then l would be 3280. Considering all these numbers and the complexity of fMF, the total number of operations would be 3.4×10^{13}, which is prohibitive. In this paper, we propose a new method (LAL-FDT), which for the same dataset, needs orders of magnitude fewer (4.58×10^9) operations.

The complexity becomes even larger when we note that in (Zhou et al, 2011) decision trees are not built only once. After building decision trees in one iteration, item features are updated and again decision trees are built using the features of the updated item. This is repeated until convergence. Although (Zhou et al, 2011) conduct experiments on the Netflix data set, unfortunately the authors do not report the running time of their algorithm. Moreover, in addition to time complexity, the required memory to store decision trees is also large, since we have to store user features of all nodes in order to update item features after building the tree.

(Karimi et al, 2013) improved (Golbandi et al, 2011) in two ways. First, it proposed the Most Popular Sampling (MPS) method to increase the speed of the tree construction. Second, it developed a new algorithm to build decision trees, which is called Factorized Decision Trees (FDT). (Karimi, 2014) summarized several active learning methods for recommender systems.

2.3 Other Approaches

Active learning has also been applied for the new-system problem (Boutilier et al, 2003; Rubens and Sugiyama, 2007; Sutherland et al, 2013; Rish and Tesauro, 2008) and the new-item problem (Park and Chu, 2009; Deodhar et al, 2009; Huang, 2007). Also, some works combine the new-system and the new-user problems, because they assume that, when new users enter the recommender system, there are not yet many active users, since the system has not been operating for a long time (Elahi et al, 2014, 2013, 2012).

Furthermore, it is worth mentioning that in addition to active learning, there are other approaches that deal with the new-user problem:

– **Implicit Feedback.** (Zhang et al, 2009; Zigoris, 2006) leverage implicit feedback, such as search keywords or user clicks to learn new user preferences.
– **Content-based recommendation** (Gantner et al, 2010; Gunawardana and Meek, 2008) combine content-based attributes with collaborative filtering.
– **Demographic information.** (Safoury and Salah, 2013) use demographic information on new users.

In this work, we focus on the new-user problem based on an active learning approach, assuming that there are already enough active users in the system (i.e., we do not consider the new-system problem). We believe that active learning is more promising than other approaches because it relies on ratings, which are more informative about the actual preferences of users compared to metadata or demographic information. That is why collaborative filtering outperforms content-based methods, as it has been reported that even a few ratings are more valuable than metadata in movie recommender systems (Pilászy and Tikk, 2009).

3 Background

3.1 Problem Definition

Let U be the set of users, I be the set of items, and $R \subseteq \mathbb{R}$ be a (finite) set of possible ratings, e.g., $R := \{1, 2, 3, 4, 5\}$. Let $R^+ := R \cup \{.\}$ with an additional symbol for a missing value. The triple $(u, i, r) \in U \times I \times R$ denotes the rating r of user u for item i.

For a data set $\mathcal{D} \subseteq U \times I \times R$ denote the set of all users occurring in $\mathcal{D}$ by:

$$U(\mathcal{D}) := \{u \in U \mid (u, i, r) \in \mathcal{D}\}$$

Subsets of the form $E \subseteq I \times R$ are called user profiles. The profile of user u in $\mathcal{D}$ is denoted by:

$$\mathcal{D}_u := \{(i, r) \in I \times R \mid (u, i, r) \in \mathcal{D}\}$$

The rating of item $i \in I$ in user profile $E \subseteq I \times R$ is denoted by:

$$r(i; E) := \begin{cases} r & \text{, if } (i, r) \in E \\ . & \text{, else} \end{cases}$$

We define a questionnaire as a tree where each interior node is labeled with an item $i \in I$, each branch with a rating value $r \in R^+$ and each leaf node corresponds to a rating prediction model $\hat{r} : I \to R$. For a user profile $E \subseteq I \times R$, let $\hat{R}(E)$ denote the rating prediction model at the leaf arrived when starting at the root of the tree and iteratively from a node with label $i \in I$ proceeds to its child node with label $r(i; E)$ until a leaf node is reached.

Given

- a data set $\mathcal{D}^{\text{train}} \subseteq U \times I \times R$,
- a loss $\ell : R \times \mathbb{R} \to \mathbb{R}$, and
- a maximal number of queries N,

the active learning for the new-user problem in recommender systems is to find a questionnaire $\hat{R}$ of maximal depth N s.t. for another data set $\mathcal{D}^{\text{test}} \subseteq U \times I \times R$ (sampled from the same distribution, not being used during training, and with non-overlapping users, i.e., $U(\mathcal{D}^{\text{train}}) \cap U(\mathcal{D}^{\text{test}}) = \emptyset$) the average loss is minimal.

Users in $\mathcal{D}^{\text{test}}$ are supposed to be new users. For each $u \in \mathcal{D}^{\text{test}}$, $\mathcal{D}_u$ is split into $\mathcal{D}_u^{\text{pool}}$ (pool data) and $\mathcal{D}_u^{\text{test}}$ (test data). $\mathcal{D}_u^{\text{pool}}$ is used to find the prediction model $\hat{R}(\mathcal{D}_u^{\text{pool}})$ at the leaf node and $\mathcal{D}_u^{\text{test}}$ is used to evaluate it. $\mathcal{D}_u^{\text{pool}}$ should also contain items with missing value, so that:

$$\mathcal{D}_u^{\text{pool}} = \mathcal{D}_u^{\text{pool}} \cup \{(u, i, .) | i \in I, i \notin \mathcal{D}_u^{\text{pool}}\}$$

The total loss is the loss over all test users:

$$\ell(\mathcal{D}^{\text{test}}; \hat{R}) := \frac{1}{|\mathcal{D}^{\text{test}}|} \sum_{u \in U(\mathcal{D}^{\text{test}})} \sum_{(i,r) \in \mathcal{D}_u^{\text{test}}} \ell(r, \hat{R}(\mathcal{D}_u^{\text{pool}})(i)) \quad (3)$$

What we call decision tree or questionnaire tree here is a multivariate regression tree. Other names could be used as well, such as 'rating prediction tree' or 'recommendation tree'.

3.2 Matrix Factorization

Matrix Factorization (MF) is a model for approximating the true, unobserved ratings-matrix R by $\hat{R} \in \mathbb{R}^{|U| \times |I|}$. It maps both users and items to a latent space of dimensionality k. In this latent space, each item i is represented by a vector $h_i \in \mathbb{R}^k$. In the same way, each user u is represented with a vector $w_u \in \mathbb{R}^k$.

The dot product $h_i^T w_u$ captures the interaction between user u and item i, which indicates how interesting is item i for user u. However, the actual behavior of users during rating is not fully explained by this interaction and the user and item bias should also be taken into account. These biases are effects associated with either users or items, independent of any interactions. For example, some users tend to give high rating to many items or some items tend to receive high ratings from many users.

By considering the user and item bias, the predicted rating is computed as follows (Koren et al, 2009):

$$\hat{r}_{ui} = \mu + b_i + b_u + h_i^T w_u \quad (4)$$

where μ is the global average, b_i is the item bias and b_u is the user bias.

Figure 1 shows a simple example of MF for a movie recommendation scenario. There are two latent dimensions. The first dimension measures female-versus male-oriented characteristic and the second dimension deals with serious versus escapist aspect. In general, the predicted ratings of users for items is proportional to their distances in this latent space. For example, we expect that Gus gives a high rating to "Dumb and Dumber" and a low rating to "The Color Purple". This means that "Dumb and Dumber" is a good recommendation for Guss and "The Color Purple" is a bad recommendation.

The major challenge is the computation of the mapping of each item and user to the factors represented by the vectors $h_i, w_u \in R^k$. The mapping is done by minimizing the following squared error (Koren, 2008):

$$Opt(\mathcal{D}^{\text{train}}, W, H) = \sum_{(u,i,r) \in \mathcal{D}^{\text{train}}} \Bigg((r - \mu - b_u - b_i - h_i^T w_u)^2 + \lambda(\|h_i\|^2 + \|w_u\|^2) + \gamma(b_i^2 + b_u^2) \Bigg) \quad (5)$$

where $\mathcal{D}^{train}$ is the set of the (u, i, r) triples for which the rating r of user u to item i is known, λ is the feature regularization factor, γ is the user and item bias regularization. The reason of regularization is to avoid over-fitting, which occurs

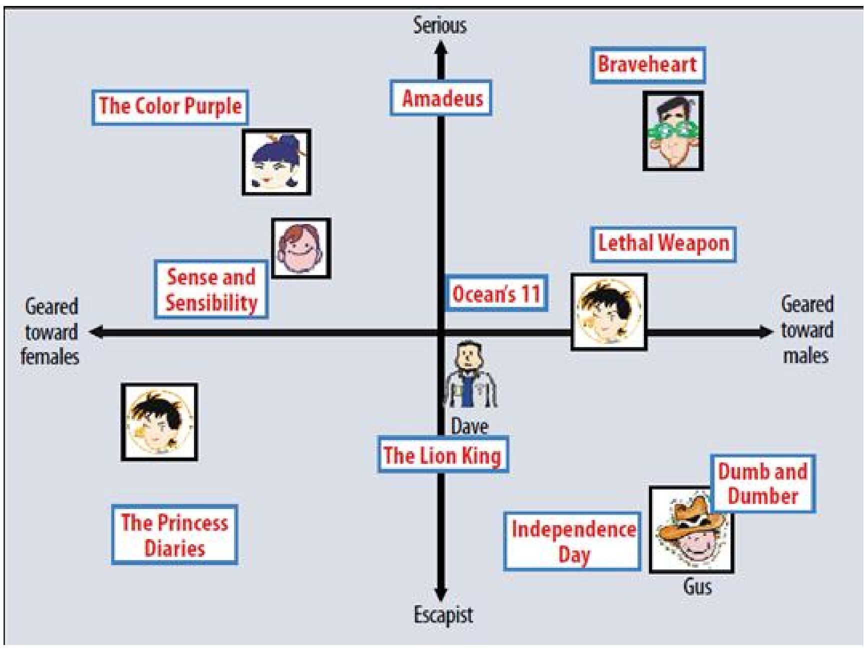

Fig. 1 A simple two dimensional latent space for movie recommendation scenario (Koren et al, 2009)

.

when a statistical model learns random error or noise instead of the underlying process. The usual method to train MF is stochastic gradient descent (Koren et al, 2009). This algorithm shuffles the ratings and then loops through all of them by picking a random triple (u, i, r) and updates the corresponding parameters in Equation 5. After each iteration (epoch), if the error computed with the loss function is smaller than ϵ, the training stops. Otherwise, it continues until reaching a maximum number L of iterations.

4 Supervised Active Learning Framework

In general, the goal of active learning is to choose those instances that will be queried query for their label, that reduce the test error as much as possible when the prediction model is retrained with the queried labels. However, directly finding the most informative query is not possible, because of two reasons: First, the test data is not available during the training, thus there is no way to measure the test error of possible queries in advance. Second, we do not know the labels of unlabeled instances before querying, so we cannot compute how a specific instance will improves the model and consequently will reduce the test error.

For the same reasons, finding the most informative query is not possible in recommender systems as well, because, first, the test data of new users are not available during the query process and, second, their answers to the queried items are not known before asking them. But in recommender systems there is additional data that does not exist in general classification problems: there are already many training (active) users. We can consider training users as (hypothetical) *new* users and solve an active learning problem for each of them. In the end, we aggregate all solved problems in order to learn how to solve the active learning problem for a real new user. As the ratings (i.e., labels) of existing users are known and are used to find informative queries to new users, the proposed framework is a *supervised active learning framework.* Moreover, aggregation acts like a post-learning phase over all solved active learning problems, so we call the proposed framework *Learning Active Learning (LAL).*

The reason that we can solve active learning problems for training users is that their ratings are known. The ratings of each user are split into pool data and validation data. The validation data is used to measure the test error of candidate items. For each item there are two errors: the error before adding the item to the train data and the error afterward. The difference between these errors reflects how informative the item is, i.e., after the item is queried and its rating is added to the training data, we can measure how effective would it be to improve the recommendation accuracy. Note that the pool data consists of all items of the data set, including those that have been rated by the user and those with missing rating. Items with missing ratings have no impact on the validation of error, therefore models trained with or without them do not differ. Algorithm 1 describes the general framework of LAL.

Algorithm 1 The General Algorithm of Learning Active Learning (LAL)

```
Input: $\mathcal{D}^{\text{train}}$
Output: $i^*$
 1: for $u \in U(\mathcal{D}^{\text{train}})$ do
 2:     split $\mathcal{D}_u^{\text{train}}$ into $\mathcal{D}_u^{\text{pool}}$ and $\mathcal{D}_u^{validation}$
 3:     $\mathcal{D}_u^{\text{pool}} = \mathcal{D}_u^{\text{pool}} \cup \{(u, i, .) | i \in I, i \notin \mathcal{D}_u^{\text{pool}}\}$
 4:     $RMSE_u^1$ = initial error of user $u$ on $\mathcal{D}_u^{validation}$
 5:     for $i \in \mathcal{D}_u^{\text{pool}}$ do
 6:         retrain the prediction model with $r_{ui}$
 7:         $RMSE_u^2$= new error of user $u$ on $\mathcal{D}_u^{validation}$
 8:         $\Delta_{ui} = RMSE_u^1 - RMSE_u^2$
 9:     end for
10: end for
11: $\bar{\delta}$ = aggregate all $\Delta_{ui}$
12: $i^* = \text{argmax}_i \, \bar{\delta}_i$
```

To compute the effect of the candidate item i on the validation error, first we measure the initial error of user u, which is denoted as $RMSE_u^1$. Then the prediction model is retrained with the rating of the candidate item i and the resulting error (denoted as $RMSE_u^2$) is measured. The difference between the second and the first validation errors indicates how much the candidate item is informative. We call this difference Δ. Another possibility is to train the model with all pool data excluding the candidate item and measure $RMSE_u^1$. Then we train again the model with all pool data including the candidate item and measure $RMSE_u^2$. However, this method is slower because it needs to be trained twice.

After finding the informativeness of all items for all training users, we need to aggregate the results in order to find the best query for real new users. There are several possibilities for such an aggregation. For example, we can compute the average Δ of each item over all users:

$$\bar{\delta}_i = \frac{1}{|U|} \sum_{u \in U} \Delta_{ui}$$

where $\bar{\delta}_i$ is the average error reduction of item i. The best item i^* is the item with the maximum $\bar{\delta}_{i^*}$. Another possibility is to count how many times an item has been the best item in each separate active learning problem for training users and choose the item with the maximum occurrence. Figure 3 provides a schematic view of LAL steps.

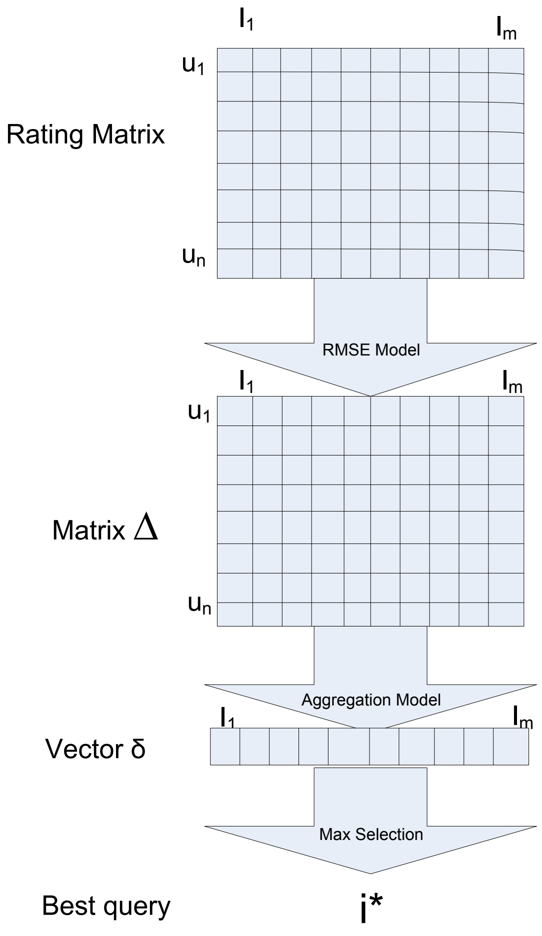

Fig. 2 A schematic view of LAL steps

Algorithm 1 identifies only the single best query. If N queries need to be asked, we need to find N items. A trivial solution would be to rank items according to their δ_i in a descending order fashion and then pick the top-N items. However, recent research indicates that selecting a fixed set of items upfront and asking them one by one, is not a good strategy (Golbandi et al, 2011; Rashid et al, 2008). Instead, it could be better to adapt the interview process according to the new users' answers to the previous queries. This means that the queries are selected while new users are being interviewed. However, switching to an adaptive approach requires a considerable computational effort. The reason is that, in each step, given the provided ratings by the new user so far, we have to execute again Algorithm 1, which is time consuming and not applicable. Note that new users are not willing to wait for a long time to be asked for the next queries, so an active-learning method should be fast. Therefore, we need a method that is both adaptive and fast.

To address the aforementioned challegne, we consider a model that is learned offline, i.e., *before* new users start the interview process. When the interview starts, given the ratings that the new user has provided so far, the model quickly finds the next best query. To capture the adaptive aspect of the model, decision trees, as they have been proposed by Golbandi et al (2011), are appropriate. So, we also use decision trees for this purpose, to build an adaptive and fast learning active learning method. In fact, the required time to find the next query requires only a simple comparison to find the child node to which the new user should be moved according to the rating this user provides about the queried item. As already explained in section 2, Golbandi et al (2011) opts for a 3-way split. However, it can be expected that a more fined-grained split will improve the accuracy, since it distinguishes users' tastes more precisely (Karimi et al, 2014). Therefore, in this paper, given the ratings of the data set is from 1 to 5, we use 6-way splits. Specifically, one child node per rating and one child node for the "unknown" answer.

Algorithm 1 checks all items to find the best query. The best query is the item that will result in the maximum reduction in error. To compute the error, we need a model for rating prediction. There are many models for rating prediction, such as global average, item average, and user average, which are simple, and more advanced models such as MF. Although MF is the state-of-the-art method for rating prediction, we do not use it for building the decision tree. The reason is that there are many nodes in the decision tree and in each node there are many candidate items that must be checked. Training an MF model for each item-node combination would need prohibitive computational time, which makes the learning algorithm intractable. For this reason, following Golbandi et al (2011), we use the item average. Namely, predictions in a node are computed as the item average over the ratings of users associated with this node. Although item average is simple, it is fast and suitable for large-scale recommender systems.

Algorithm 2 describes the updated algorithm of LAL based on decision trees, which is used to select N queries. Suppose that we are at node t and we are going to ask the q-th query, which must be selected from D^{pool}. For performance reasons, $\mathcal{D}^{\text{train}}$ is already split into $\mathcal{D}_u^{pool}$ and $\mathcal{D}_u^{validation}$ for all training users u and the resulting sets are accessible as global variables in ConstructDecisionTree function. $\mathcal{D}_u^{pool}$ includes all items of the data set. $\hat{r}_t$ is a vector that stores rating predictions in node t for all items using the item average method (Equation 1). It is computed based on $\mathcal{D}_u^{pool}$ of all users in node t. Compared to Algorithm 1, the main change concerns the computation of $RMSE_u^1$ and $RMSE_u^2$ based on decision trees. $RMSE_u^1$ is the error of user u in the current node t that is computed based on rating predictions $\hat{r}_t$ and $\mathcal{D}_u^{validation}$. Each pool (candidate) item splits the node into 6 child nodes. In the child node v, the rating predictions are computed based on $\mathcal{D}_u^{pool}$ of all users in node v. To compute the $RMSE_u^2$, first we need to find the child node v to which user u has moved. Then, $RMSE_u^2$ is computed based on $\hat{r}_v$. Note that like (Golbandi et al, 2011), the ratings in "Unknown" child node are computed based on the collected statistics from other child nodes (1 to 5). Finally, Δ_{ui} is computed, which is the difference between the original $RMSE_u^1$ and the resulting $RMSE_u^2$. After finding all Δ_{ui}, we aggregate them by using the average function. The item with the maximum δ_i value is selected to be queried. After finding the best query i^*, we need to go on to find the next best queries. So, we create child nodes using i^* and recursively call the ConstructDecisionTree function to split these nodes. Note that in order to avoid selecting one item twice, i^* is removed from the pool items of the child nodes.

Algorithm 2 LAL based on decision trees

```
ConstructDecisionTree
Input: $U^t$, $D^{pool}$, $q$,$\hat{r}_t$
Output: $i^*$
1: for $u \in U^t$ do
2:     compute $RMSE_u^1$ based on $\hat{r}_t$
3: end for
4: for $i \in D^{pool}$ do
5:     split $U^t$ into 6 child nodes according to their ratings to item $i$
6:     for each child node $v$ do
7:         compute rating predictions $\hat{r}_v$
8:     end for
9:     for $u \in U^t$ do
10:        find the child node $v$ that user $u$ has moved to
11:        compute $RMSE_u^2$ based on $\hat{r}_v$
12:        $\Delta_{ui} = RMSE_u^1 - RMSE_u^2$
13:    end for
14: end for
15: $\bar{\delta}$ = aggregate all $\Delta_{ui}$
16: $i^* = \text{argmax}_i\ \bar{\delta}_i$
17: if $q < N$ and $\bar{\Delta}_{i^*} \geq 0$ then
18:     create 6 child nodes based on the selected item $i^*$
19:     $D_{new}^{pool} = D^{pool} \backslash \{i^*\}$
20:     for child node $v$ do
21:         recursively call ConstructDecisionTree ($U^v$,$D_{new}^{pool}, q+1, \hat{r}_v$)
22:     end for
23: end if
```

In Algorithm 2, there are two stopping criteria. The first checks whether the number q of queries that we have already asked so far is equal to the maximum number N of queries that we are supposed to ask. The second criterion is about δ_{i^*}.

In most of the cases, δ_{i^*} is expected to be positive. This means that the splitting of nodes will result in a more coherent cluster of users, which consequently improves the recommendation accuracy. However, sometimes this does not happen and there is no item that, after splitting the current node with it, would improve the accuracy of the predictions of the current node. In such cases, δ_{i^*} is negative. One reason for this phenomenon is that, as we go down into the deep layers of the decision trees, the number of associated users of nodes decreases. Therefore, the ratings in such nodes are predicted with less training data, which obviously adversely affects the accuracy. Although the predictions are still regularized towards the predictions in the parent node, this regularization might not be sufficient to compensate for the effect of less training data in such nodes. In the case of negative δ_{i^*}, the current node is not split any further.

4.1 LAL vs. Bootstrapping

LAL uses decision trees to adapt the preference elicitation process to the new user's answers. As (Golbandi et al, 2011) is also based on decision trees, there are similarities between these two approaches. In this section, we would like to clarify the similarities and also highlight the novelties of our approach.

We first revisit Bootstrapping from the point of view of the proposed LAL method. For each candidate item, we want to know how good this item would be in order to be selected for querying the new user. The reasons is that its rating is not known upfront. However, there are already many training users and their ratings are available. Thus, instead of trying to find out how good the item is for the new user, we investigate how good this item is for training users. The training users are divided into three groups: those who *like* it, *dislike* it and do not know it (*unknown*). We treat the users of each group as a single user and compute the same rating predictions for all of them. The rating predictions are simply the item average. After computing the predictions, we need to evaluate them to find out how accurate the predictions are. The evaluation is done individually in each group. For this purpose, all ratings of the corresponding users' group are united and are considered as validation data. The validation error of each group is the square error over all validation data of the corresponding group. Finally, the three resulting validation errors are summed up to compute the total validation error of the candidate item. The item that has the minimum error is selected for query.

When comparing Bootstrapping to LAL conceptually, we can consider that Bootstrapping also builds an error matrix E, which is similar to the matrix Δ. However, matrix E is filled out in a column-wise fashion, instead of the row-wise manner in Δ. Also, the aggregation function is a summation operator, which sums up over all errors in a column to find the total error of each item. In the end, the item with the minimum error is the best item to query. In fact, both approaches aim to exploit the additional information from existing users to find informative queries. However, they look at this information from different angles. LAL starts from a training user and treats the user as a hypothetical new user and then computes the informativeness of each candidate item for this user. By calculating the informativeness of items for training users, the system can estimate the informativeness of items for test users. On the other hand, Bootstrapping starts at an item and computes the error of the item for all training users, which gives an estimation for the error of test users. In addition to this conceptual difference, there are three specific points that makes the LAL a suitable approach:

1. In Bootstrapping the square error is used, whereas LAL is based on RMSE. In our experimental study we use RMSE for the evaluation of recommendation accuracy. For this reason it is meaningful to use the same measure during optimization.
2. In LAL, the informativeness of each item is a relative value, which is the difference between the current error and the new error after querying the item. But in Bootstrapping, the informativeness is an absolute value, computed only based on the new error. The relative informativeness is expected to be more precise, because it provides more information on the effectiveness of the items to reduce the error.
3. In Bootstrapping, training data and validation data are the same. Namely, the same data that has been used to predict ratings (using the item average method) is also used for evaluation. However, in LAL, the validation data does not appear in the pool data, so it is not used to compute predictions, which is expected to lead to a better generalization.

4.2 Further Improvements in LAL

(Karimi et al, 2013) reported that split items in Bootstrapping are the most popular items among the associated users of nodes and the rest of items are irrelevant. We called this method Most Popular Sampling (MPS). The results on the Netflix dataset indicated that, by sampling the 200 most popular items at each node, we can save a considerable running time without harming accuracy. In this paper, we also use MPS with sampling size equal to 200.

Another finding of (Karimi et al, 2013) is that the rating predictions in the leaf nodes of decision trees can be improved by MF. We called this approach Factorized Decision Trees (FDT). Again in this paper, we use FDT to improve the rating predictions of decision trees built by LAL. In this way, we can benefit from the right strategy for active learning and also the right prediction model for rating prediction. In the experiment, we report the results of LAL on two prediction models: item average and MF.

4.3 Complexity Analysis of LAL

In order to analyze the complexity of LAL, we follow the same procedure as in Bootstrapping (Golbandi et al, 2011). Specifically, we find the complexity of splitting one node, then we extend it to one level, and finally to the entire decision tree.

The core part of LAL in Algorithm 2 consists of lines 4 to 14, where the best split item is found. Compared to Bootstrapping, this part has three differences. In Bootstrapping, nodes are split into three child nodes, the error is calculated using the training data of associated users U^t, and the best split item is the item that reduces the error of child nodes as much as possible. Different from Bootstrapping, LAL splits nodes into six child nodes, the error is computed using the validation data of associated users U^t, and most importantly, the best split item is the item that maximizes the amount of error reduction over all users. Despite these differences, the core part of LAL and Bootstrapping have the same time complexity. On the other hand, the complexity of other parts of LAL, before and after the core part, is dominated by the complexity of this part. Therefore, the complexity of splitting node t in LAL is $O(\sum\limits_{u \in U^t} |\mathcal{D}_u|^2)$. Like Bootstrapping, as at each level of the tree a user belongs to only a single node, the time complexity of computing a single tree level is $O(\sum\limits_{u \in U} |\mathcal{D}_u|^2)$ or shorter $O(|\mathcal{D}|^2)$. Finally, given that N queries are going to be asked, the tree has N levels, thus the complexity of building the entire tree is $O(N \cdot |\mathcal{D}|^2)$. If we use FDT to update the rating predictions at leaf nodes, we should also consider the overhead of training an MF model. In this case, the total complexity of LAL will be $O(N \cdot |\mathcal{D}|^2 + |\mathcal{D}| \cdot k \cdot L)$.

5 Experimental Evaluation

In this section, we experimentally examine the performance of LAL. Our objective is to investigate the accuracy of rating prediction with respect to the number of queries asked to new users. It is important to obtain improvement in accuracy after a small number of queries, since users are generally reluctant to answer many of such queries.

5.1 Experimental Setup

The main challenge in applying active learning for recommender systems is that users are not willing to answer many queries in order to rate the queried items. For this reason, we report the performance of all examined methods in terms of prediction error (RMSE) versus the number of queried items, which is simply denoted as the *number of queries*. The RMSE of user u is computed as follows:

$$RMSE_u = \sqrt{\frac{1}{|\mathcal{D}_u^{test}|} \sum_{(i,r) \in \mathcal{D}_u^{test}} (r - \hat{r}_{ui})^2} \tag{6}$$

where $\mathcal{D}_u^{test}$ is the set of the test items of user u, $\hat{r}_{ui}$ is the predicted rating of user u for item i, and r is the true (actual) rating. Thus, we examine the problem of selecting at each step, the item for which each the test user u will be queried to provide a rating. Note that test users are acting as new users of the system. The reported RMSE is the average over all test users.

RMSE is a typical error measure for rating prediction in recommender systems, which is also used in the baseline (Golbandi et al, 2011) and in the Netflix prize. Moreover, as the algorithm of LAL is based on optimizing RMSE, we should also evaluate it using the same optimization criterion. However, there are some considerations concerning the use for evaluating the performance of recommender systems. During the Netflix prize it has been reported that even 1% lift in RMSE leads to a significant difference in the ranking of the "top-10" most recommended movies for a user (Koren, 2007). Therefore, we also evaluate our approach based on RMSE.

To reproduce the reported results of this paper, the way that the parameters are initialized should be taken into account. The parameters of MF are initialized randomly with a normal distribution of $N(0, 001)$. The random seed is set and then a sequence of random numbers are generated to initialize user features, item features, user bias, item bias, and finally to shuffle the dataset and choose a random instance (u, i, r) for the stochastic gradient descent.

In our experiment, each new user is being asked 6 queries. We pick Bootstraping (Golbandi et al, 2011) and our own previous work (Karimi et al, 2013), called FDT as baselines for the comparison. Bootstrapping was implemented by calibrating with the same hyper-parameters as reported in (Golbandi et al, 2011). After this calibration step, we changed one of the hyper-parameters in our experiments. In the original Bootstrapping (Golbandi et al, 2011), nodes in which the number of ratings is less than a predefined threshold are not expand, because the expected reliable refinement that could be gained is negligible. For this purpose the β threshold of Bootstraping was set in (Golbandi et al, 2011) equal to 200000. The motivation for having this threshold is to save runtime. In our experiments, however, we set β to zero, because MPS is already able to save runtime and there is no need to stop learning. The results show that this setting is significantly beneficial. For instance, when $\beta = 200000$, RMSE is 0.971 after 5 queries, whereas for $\beta = 0$ RMSE can be reduced to 0.958.

As the performance of baselines has been experimentally investigated based on the Netflix dataset, we also run our experiments on this dataset. Netflix includes 100,480,507 ratings of 480,189 users to 17,770 movies. The dataset is already split into train and test datasets. However, this split is not suitable for cold-start evaluation protocol, since users in the training and test sets are the same. As test users are considered as new users, they should not already appear in the training set. Therefore, we split all users into two disjoint subsets, training users and test users, containing 75% and 25%, respectively. The tree is learned based on the ratings of the training users in the Netflix training data. Ratings of the training users in the Netflix test data set are used as the validation data in LAL and also for finding hyper-parameters in MF. Users in the test set are assumed to be new users. The ratings of the test users in the Netflix training data set are used to generate the user responses in the interview process (pool data). To evaluate the performance after each query, the ratings of test users in the Netflix test data are used.

We conduct five-fold cross validation and report the average $RMSE$. In our experiments, the variances of the results on different folds were very small (around 0.000003). We also ran a t-test and measured p-values in all queries. The p-values were very small (in the range of 0.0003 to 0.0005). The reason is that the data set is big. It can be assumed that, due to the same reason, (Golbandi et al, 2011) did not report results on statistical significance.

We report the performance of LAL based on two prediction models. LAL-Boot, like Bootstrapping, leverages item average for rating prediction. On the other hand, LAL-FDT, like FDT, exploits MF to update rating predictions at leaf nodes after constructing decision trees.

We also compare our work to four simple baselines. The goal of this comparison is to assess the difficulty of the new-user problem. These four baselines are:

- **Random:** At each node, the split item is selected randomly.

- **Local Most Popular (LMP):** At each node, the most popular item according to the users associated with the node is selected.
- **Global Most Popular (GMP):** First, the s most popular items are found based on all ratings available in the dataset. Then we start building questionnaire trees. All the nodes that are at level l are expanded using the l-th most popular item. In this way, the dynamic aspect of questionnaire trees is omitted and all new users, regardless of their responses to the queries, receive the same questions.
- **Entropy and Popularity (Ent*Pop):** For all candidate items, the entropy of ratings that are within a node is computed and then is multiplied with the log of the number of ratings received by the item. Finally, the item with the highest value is selected. Note that the entropy is local (within a node) but the popularity is global (in the whole data set).

Similarly to (Golbandi et al, 2011), the prediction model of these baselines are item average.

5.2 Results

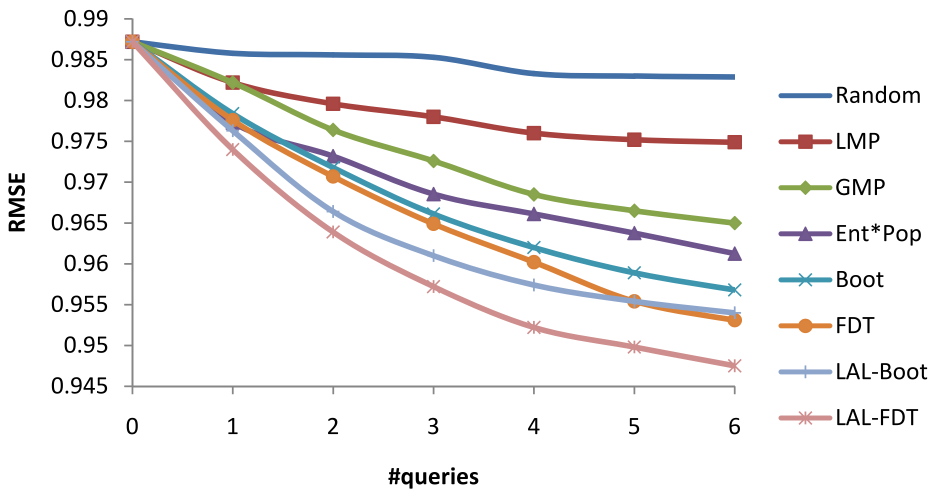

Fig. 3 RMSE results of four simple baselines, LAL-FDT, LAL-Boot, FDT and Bootstrapping

Figure 3 depicts the results of four simple baselines, LAL-FDT, LAL-Boot, FDT and Bootstrapping. As these results show, random item selection performs very badly and gains almost nothing after 6 queries. LMP does not work well either. Among the four simple baselines, GMP and Ent*Pop are the best, although their performance is still worse than Bootstrapping. Table 1 shows some statistics that can explain these results. The table shows the probabilities of receiving different responses from new users by each method. The main reason that random selection does not perform well is that it chooses items that will not be rated by new users. The probability that random selection receives a rating is less that 0.01. When the new user does not rate the queried item, that new user is moved to the unknown child node. As the predictions in the unknown child node do not significantly differ from the predictions at the current node, this strategy is not able to improve the accuracy of predictions. Remember the premise that test users and training users have the same distributions. If test (new) users do not know the split item, training users do not know it either. Therefore, decision trees which are built using training users with the random selection strategy are very imbalanced. This means that almost all users of the current node are moved to the unknown child node and consequently the predictions at the current node and the child nodes would be almost the same (Karimi et al, 2013). LMP and GMP receive more ratings compared to the random selection, that is why their performance also improves in Figure 3. Interestingly, Ent*Pop performs worse than GMP in selecting familiar items (i.e., items that users know and for which they will provide a rating), but its recommendation accuracy is better.[3] The reason is that, the selection of familiar items is not the only factor that can improve the accuracy. In addition to that, the queried items must be informative, i.e., they must be effective to reveal the preferences of the new users. This is stated in (Rashid et al, 2002) with the following illustrating example: “Popular movies may be widely liked; if this is true, then their ratings carry little information. If everyone likes Titanic, and I say I like it too, what can the system learn from that”. On the other hand, disputing items are more informative because some users like them and some users dislike them, therefore it would be informative for the system to know the opinion of the new users on them. Finally, compared to Ent*Pop, Bootstrapping and FDT have about the same chance to select the known items, although their performance (in terms of recommendation accuracy) is higher because it does not exploit any heuristic and directly computes the error and chooses the item that leads to the minimum error.

Table 1 The probability that the new user likes the queried item (p_{like}), dislikes it ($p_{dislike}$), or does not rate it ($p_{unknown}$) for different active learning methods.

Method	p_{like}	$p_{dislike}$	$p_{unknown}$
Random	0.004	0.003	0.993
LMP	0.18	0.16	0.66
GMP	0.26	0.17	0.57
Ent*Pop	0.15	0.19	0.66
Bootstrapping and FDT	0.18	0.15	0.67

In order to compare the proposed method (LAL-FDT) with other approaches, first we compare it against FDT to find the right query selection strategy, given that the same prediction model (MF) is used. Clearly, LAL-FDT outperforms FDT for all examined number of queries. The same fact is also observed when we compare LAL-Boot to Bootstrapping. The reason is that the proposed criterion

[3] We also examined a pure entropy approach that performed worse than GMP and for this reason we omit the presentation of its results.

of LAL outperforms the criterion of Bootstrapping to select queries, regardless of the rating prediction model. However, we have to mention that the premise for the improved performance of LAL is that the test data (new users) and training data (existing users) follow the same distribution. Therefore, when LAL treats training users as hypothetical new users and finds the best queries for them, the same items would also represent the best queries to new users as well. Moreover, LAL uses relative RMSE to find the informativeness of items, which is a better measure compared to the absolute square error in Bootstrapping. Finally, in LAL the errors are computed based on validation data, and as this data do not appear in the training data, the obtained errors would be closer to the test error (i.e., better generalization), which leads to selecting more informative items.

As Figure 3 shows, going beyond query 6 is not beneficial for LAL-FDT. For FDT and Bootstrapping, this convergence happens after 8 and 10 queries, respectively (Karimi et al, 2013) (these results are omitted). In general, decision trees would not help too much after a specific level where the number of nodes exceeds a threshold and this threshold depends on the query selection strategy and the prediction models (MF or item average). However, it has to be remind that the goal of the proposed approach is to reach this convergence by using a small number queries, because new users are not willing to answer many questions.

After 5 queries, LAL-Boot converges to FDT and it even starts to become worse with the sixth query. This happens because, as we go down to the deeper layers of decision trees, the number of associated users of nodes drops. Therefore, the ratings in such nodes are predicted with less training data, which adversely affects accuracy. Although the predictions are still regularized towards the predictions in the parent node, this regularization might not be sufficient to compensate for the effect of less training data in such nodes. However, FDT does not suffer from this problem, because it does not use hierarchical regularization, instead it exploits typical ℓ_2 regularization. In this way, LAL-FDT takes advantage of both LAL-Boot and FDT. It enjoys a right strategy to select a query and a right prediction model to benefit from its response.

The report improvement achieved in this reported result is substantial, especially when compared to improvement reported in related work, such as (Jin and Si, 2004; Harpale and Yang, 2008) (even for smaller datasets), or in similar problems, such as this recent paper (Chen et al, 2013). The reason is that even a small lift in RMSE leads to significantly better overall performance (w.r.t. to user experience) of the recommender system (Koren, 2007).

Table 2 summarizes the RMSE (after 6 queries) and the complexity of LAL-FDT (proposed method), LAL-Boot, FDT, Bootstrapping, and Ent*Pop as the best simple baseline. LAL-FDT and FDT have the same complexity, while the accuracy of LAL-FDT is higher. LAL-FDT is more complicated than Bootstrapping, because it needs to train an MF model. In our experiments, training a MF model takes around 3 hours, which after considering the gained improvement, pays off.

A grid search methodology was followed to find hyper-parameters for LAL-MF. Grid search is an exhaustive search in a manually specified subset of the hyperparameter space of a learning algorithm. In our experiments, the range of α and λ were $[0.0001, 0.01]$ and $[0.001, 0.03]$ respectively. The hyper-parameters of LAL-FDT and FDT in all levels are reported in table 3 and 4, respectively. In our experiments, varying k did not change the results, so we fixed it to 70. Also, the best value of γ was 0.0001 for all levels.

Table 2 The RMSE (after 6 queries) vs. the complexity of the proposed method (LAL-FDT) and the baselines.

Method	RMSE	Complexity
Ent*Pop	0.9612	$O(N.\|\mathcal{D}\|)$
Boot	0.9568	$O(N.\|\mathcal{D}\|^2)$
LAL-Boot	0.9540	$O(N.\|\mathcal{D}\|^2)$
FDT	0.9531	$O(N.\|\mathcal{D}\|^2 + \|\mathcal{D}\|.k.L)$
LAL-FDT	0.9475	$O(N.\|\mathcal{D}\|^2 + \|\mathcal{D}\|.k.L)$

Table 3 Hyper-parameters of LAL-FDT in all levels

level	α	λ
1	0.001	0.001
2	0.0013	0.005
3	0.001	0.003
4	0.001	0.003
5	0.0004	0.009
6	0.0006	0.017

Table 4 Hyper-parameters of FDT in all levels

level	α	λ
1	0.0013	0.001
2	0.0013	0.001
3	0.001	0.001
4	0.0012	0.003
5	0.0012	0.003
6	0.0013	0.01

5.3 Decision trees versus online updating

Decision trees enable us to develop adaptive and fast active learning methods. Another approach to develop a fast active learning method is to use online updating (Rendle and Schmidt-Thieme, 2008). In online updating, when a new rating from the new user is received, only latent features of the new user in MF model are updated and the rest of features, including item features and features of the training users, are not touched. Although online updating is fast, its accuracy is much worse than decision trees.

Table 5 shows the RMSE of applying most popular item selection, given online updating is used. Compared to the most popular active learning for decision trees (Figure 3), online updating loses with a large margin. The initial error in online updating is 1.0560, which is much larger than the item average (0.9872). One could argue that the initial error in online updating is based on the random initialization of user features since new users have not given any ratings before the first query. However, this gap cannot be compensated in the next queries. Even the RMSE of online updating after five queries is still larger than the initial RMSE of decision trees. Therefore, we can conclude that online updating and in general methods that solely rely on the provided ratings by new user for rating predictions are not suitable approaches for the new-user problem in recommender systems because new users will eventually provide a few ratings, which is not enough to train an accurate model. Due to this fact, the chosen baseline (Golbandi et al, 2011), is stronger than other related works, such as (Lee et al, 2013; Harpale and Yang, 2008; Jin and Si, 2004). Although the accuracy of item average in decision trees is higher than the online updating, it does not provide latent features of new users. Latent features are necessary, if we want to fold-in new users in an MF model that already exist in the recommender system for active users. This issue reveals another advantage of FDT over Bootstrapping. In FDT, the latent features of new users have already been learnt and just need to be copied in the MF model. Moreover, its accuracy is also higher than Bootstrapping. In fact, the FDT is a new approach for online updating in which, given a new user with a few ratings, we pick the training users with the same ratings and then use all of their ratings to learn the new user's latent features. A more complete comparison between decision trees and online updating can be found in (Karimi et al, 2014).

Table 5 The accuracy of the most popular active learning for MF in the first three queries. MF is retrained using the online updating technique.

ini. query	query 1	query 2	query 3
1.056	1.0395	1.0295	1.0234

5.4 Limitations

In general, applying active learning to cold-start problem requires the cooperation of users to provide ratings to the queried items. Usually, users are willing to spend a little time and answer the queries in order to get better recommendations afterwards. However, if users are reluctant to do so, we need to switch to other approaches for the cold-start problem, such as implicit feedback (Zigoris, 2006; Zhang et al, 2009), Content-based recommendation (Gantner et al, 2010; Gunawardana and Meek, 2008), or demographic information (Safoury and Salah, 2013).

The willingness of users to provide ratings is not adequate to guarantee the success of the user preference elicitation process. If users are asked to provide ratings to items that they do not know, they will not be able to rate the items. In our setting, this corresponds to the answer denoted as "unknown". In this case, no information is gained about a user's taste and, therefore, asking or not asking the queries will result in the same accuracy in terms of modeling the preferences of this user. In this paper, this issue has been addressed by limiting the queries to the most popular items. Items that have been rated by many training users, are more probable to receive ratings from new users as well. However, the effectiveness of this heuristic in other types of data sets needs to be investigated. For data sets like books, one could expect the same outcome, since most popular books would also be known to many people. However, in other recommendation scenario (e.g., for cloth items), it might be less straightforward to identify which types of items are most popular. In such cases, showing a description of items to users could (e.g., a picture) can be helpful, because even if users have not already experienced the items, they would be able to rate them.

In this paper, we assume that the train data set is large. This means that the recommender system has already many existing users and then the new users join the system. Considering many popular recommender systems, such as Amazon, Netflix, or eBay, this assumption is realistic, because those systems have already many users. Hence, this assumption is also made in the baseline paper (Golbandi et al, 2011). However, when the recommender system is new, the number of users is small, because the systems is in the early stage and has not been used by many users so far (Elahi et al, 2014, 2013, 2012). In this case, the performance of the proposed approach (LAL) might be affected. The reason is that LAL treats training users as hypothetical new users in order to learn how to solve the active learning problem for new users. Therefore, if the number of training users is small, there may be types of test users that have not been included in the training data, and therefore the system is unable to predict the right queries for them.

To make LAL adaptive to responses of new users, we have used a decision-tree based approach, as proposed by (Golbandi et al, 2011). While this model works fine for the Netflix data, it may become slower for much larger data sets (e.g., Yahoo Music that contains 717 M ratings and thus is seven times bigger than Netflix). Although MPS could significantly speed up the tree construction algorithm, relying solely on decision trees to implement LAL for such big data sets could still be challenging.

6 Summary and Future Work

In this paper, we proposed an innovative approach to apply active learning for the new-user problem in recommender systems. The main idea is to consider past users as (hypothetical) new users in order to learn the right queries to be asked to new users for active-learning purposes. Based on this framework, we investigated two different types of models: the first model is based on information about average item ratings and the second on FDT. The results on the Netflix dataset indicate that the best improvement is achieved when LAL is combined with FDT.

Principally, there are two tasks in recommender systems: rating prediction and item recommendation. In this paper, we proposed LAL for predicting rating of new users. As future work, we plan to use LAL for item recommendation, too. In this case, we have to change the objective function in LAL. Moreover, instead of RMSE, other criteria, such as F-measure, recall, and precision will be more suitable for the evaluation of this case.

LAL relies on two models. The first model is used to measure the test error of training users to generate the matrix $\varDelta^{U \times I}$, whereas the second model aggregates this matrix into a vector δ^I containing the overall error reduction of items. In this work, we used decision trees to generate matrix $\varDelta$, and δ was computed by taking the average over all observed $\varDelta_{ui}$ values for each item. There are several ways to extend our work by changing these models. We can use other models besides decision trees to generate matrix Δ . It would also be interesting to use other aggregation models, for example a counting model that counts how many times an item has been selected as the best item for training users and chooses the one with the maximum number of occurrences.

The proposed approach can also be applied in other problems in machine learning that have settings similar to recommender systems. In recommender systems, there is a matrix of users and items, and the entities are ratings given by users to items. There are other (dyadic) datasets with the similar format, e.g.:

– MIT-DP[4]: is from MIT RealityMining project. Rows represent the blue tooth devices and columns represent the persons. The entities represent the scanning activities between the devices and persons.
– NIPS-PW[5] is from the NIPS proceedings. Rows represent papers and columns represent words. The entities represent the count of the words that appear in the corresponding papers.
– CIKM-PA[6] is an author-paper graph constructed from CIKM proceedings. Rows represent the authors and columns represent the papers. The entities indicate whether the papers are written by the authors or not.

Similar to recommender systems, a new row (column) may be added to the matrix. On the other hand, it is costly to fill the entities of the new row (column). Therefore, a few columns (rows) must be selected for being labeled.

Finally, we plan to conduct an online user study to validate to results of our offline evaluations.

References

Adomavicius G, Tuzhilin A (2005) Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge and Data Engineering 17(6):734–749

Boutilier C, Zemel RS, Marlin B (2003) Active collaborative filtering. In: In Proceedings of the Nineteenth Annual Conference on Uncertainty in Artificial Intelligence, Acapulco, Mexico, pp 98–106

Burke R (2002) Hybrid recommender systems. User Modeling and User Adapted Interaction 12(4):331–370

Chen T, Li H, Yang Q, Yu Y (2013) General functional matrix factorization using gradient boosting. In: Proceedings of the 30th International Conference on Machine Learning (ICML-13), JMLR Workshop and Conference Proceedings, vol 28, pp 436–444

Cohn DA, Ghahramani Z, Jordan MI (1996) Active learning with statistical models. J Artif Int Res 4(1):129–145

Deodhar M, Ghosh J, Saar-Tsechansky M (2009) Active learning for recommender systems with multiple localized models. In: Proc. Fifth Symposium on Statistical Challenges in Electronic Commerce Research

[4] http://realitycommons.media.mit.edu/
[5] http://www.datalab.uci.edu/author-topic/NIPs.htm
[6] http://www.informatik.uni-trier.de/~ley/db/conf/cikm/

Elahi M, Ricci F, Rubens N (2012) Adapting to natural rating acquisition with combined active learning strategies. In: Proceedings of the 20th International Conference on Foundations of Intelligent Systems, Springer-Verlag, Berlin, Heidelberg, ISMIS'12, pp 254–263

Elahi M, Braunhofer M, Ricci F, Tkalcic M (2013) Personality-based active learning for collaborative filtering recommender systems. In: AI*IA, Springer, Lecture Notes in Computer Science, vol 8249, pp 360–371

Elahi M, Ricci F, Rubens N (2014) Active learning strategies for rating elicitation in collaborative filtering: A system-wide perspective. ACM Trans Intell Syst Technol 5(1):13:1–13:33

Gantner Z, Drumond L, Freudenthaler C, Rendle S, Schmidt-Thieme L (2010) Learning attribute-to-feature mappings for cold-start recommendations. In: Proceedings of the 2010 IEEE International Conference on Data Mining, IEEE Computer Society, Washington, DC, USA, ICDM '10, pp 176–185

Golbandi N, Koren Y, Lempel R (2011) Adaptive bootstrapping of recommender systems using decision trees. In: WSDM, ACM, pp 595–604

Gunawardana A, Meek C (2008) Tied boltzmann machines for cold start recommendations. In: Proceedings of the 2008 ACM Conference on Recommender Systems, ACM, New York, NY, USA, RecSys '08, pp 19–26

Harpale AS, Yang Y (2008) Personalized active learning for collaborative filtering. In: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval, ACM, pp 91–98

Huang Z (2007) Selectively acquiring ratings for product recommendation. In: Proceedings of the Ninth International Conference on Electronic Commerce, ACM, New York, NY, USA, ICEC '07, pp 379–388

Jin R, Si L (2004) A bayesian approach toward active learning for collaborative filtering. In: Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence, AUAI Press, Arlington, Virginia, United States, UAI '04, pp 278–285

Karimi R (2014) Active learning for recommender systems. KI-Künstliche Intelligenz pp 1–4

Karimi R, Freudenthaler C, Nanopoulos A, Schmidt-Thieme L (2011a) Active learning for aspect model in recommender systems. In: CIDM, Paris, France, pp 162–167

Karimi R, Freudenthaler C, Nanopoulos A, Schmidt-Thieme L (2011b) Non-myopic active learning for recommender systems based on matrix factorization. In: IRI, Las Vegas, USA, pp 299–303

Karimi R, Freudenthaler C, Nanopoulos A, Schmidt-Thieme L (2011c) Towards optimal active learning for matrix factorization in recommender systems. In: ICTAI, Florida, USA, pp 1069–1076

Karimi R, Freudenthaler C, Nanopoulos A, Schmidt-Thiemee L (2012) Exploiting the characteristics of matrix factorization for active learning in recommender systems. In: RecSys, Dublin, Ireland, pp 317–320

Karimi R, Wistuba M, Nanopoulos A, Schmidt-Thieme L (2013) Factorized decision trees for active learning in recommender systems. In: 2013 IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI), IEEE, Washington DC, USA, pp 404–411

Karimi R, Nanopoulos A, Schmidt-Thieme L (2014) Improved questionnaire trees for active learning in recommender systems. In: Proceedings of the 16th LWA

Workshops: KDML, IR and FGWM, Aachen, Germany, September 8-10, 2014., pp 34–44, URL http://ceur-ws.org/Vol-1226/paper09.pdf

Kohrs A, Mérialdo B (2001) Improving collaborative filtering for new-users by smart object selection. In: ICME 2001, International Conference on Media Futures, 8-9 May 2001, Florence, Italy, Florence, ITALY

Konstan JA, Miller BN, Maltz D, Herlocker JL, Gordon LR, Riedl J (1997) GroupLens: Applying collaborative filtering to usenet news. Communications of the ACM 40(3):77–87

Koren Y (2007) How useful is a lower rmse? http://www.netflixprize.com/community/viewtopic.php?id=828/, accessed: 15-04-2013

Koren Y (2008) Factorization meets the neighborhood: a multifaceted collaborative filtering model. In: Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, KDD '08, pp 426–434

Koren Y, Bell R, Volinsky C (2009) Matrix factorization techniques for recommender systems. Computer 42:30–37

Lee J, Jang M, Lee D, Hwang WS, Hong J, Kim SW (2013) Alleviating the sparsity in collaborative filtering using crowdsourcing. In: Workshop on Crowdsourcing and Human Computation for Recommender Systems (CrowdRec), p 5

Park ST, Chu W (2009) Pairwise preference regression for cold-start recommendation. In: Proceedings of the Third ACM Conference on Recommender Systems, ACM, New York, NY, USA, RecSys '09, pp 21–28

Pilászy I, Tikk D (2009) Recommending new movies: Even a few ratings are more valuable than metadata. In: Proceedings of the Third ACM Conference on Recommender Systems, ACM, New York, NY, USA, RecSys '09, pp 93–100

Rashid AM, Albert I, Cosley D, Lam SK, Mcnee SM, Konstan JA, Riedl J (2002) Getting to know you: Learning new user preferences in recommender systems. In: Proceedings of the International Conference on Intelligent User Interfaces, ACM Press, pp 127–134

Rashid AM, Karypis G, Riedl J (2008) Learning preferences of new users in recommender systems: an information theoretic approach. ACM, New York, NY, USA, vol 10, pp 90–100

Rendle S, Schmidt-Thieme L (2008) Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In: Proceedings of the 2008 ACM Conference on Recommender Systems, ACM, New York, NY, USA, RecSys '08, pp 251–258

Rish I, Tesauro G (2008) Active collaborative prediction with maximum margin matrix factorization. In: ISAIM

Rubens N, Sugiyama M (2007) Influence-based collaborative active learning. In: Proceedings of the 2007 ACM Conference on Recommender Systems, ACM, New York, NY, USA, RecSys '07, pp 145–148

Safoury L, Salah A (2013) Exploiting user demographic attributes for solving cold-start problem in recommender system. Lecture Notes on Software Engineering 1(3):303–307

Sutherland DJ, Póczos B, Schneider J (2013) Active learning and search on low-rank matrices. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, USA, KDD '13, pp 212–220

Tong S, Koller D (2001) Active learning for structure in bayesian networks. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 2, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, IJCAI'01, pp 863–869

Tong S, Koller D (2002) Support vector machine active learning with applications to text classification. Journal of Machine Learning Research 2:45–66

Yang B, Sun JT, Wang T, Chen Z (2009) Effective multi-label active learning for text classification. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, USA, KDD '09, pp 917–926

Zhang L, Meng X, Chen J, Xiong SC, Duan K (2009) Alleviating cold-start problem by using implicit feedback. In: ADMA, Springer, Lecture Notes in Computer Science, vol 5678, pp 763–771

Zhou K, Yang SH, Zha H (2011) Functional matrix factorizations for cold-start recommendation. In: 34th international ACM SIGIR conference on Research and development in Information Retrieval, ACM, SIGIR '11, pp 315–324

Zigoris P (2006) Bayesian adaptive user profiling with explicit & implicit feedback. In: In Conference on Information and Knowledge Mangement, ACM Press, p 397