Improved Questionnaire Trees for Active Learning in Recommender Systems

Rasoul Karimi[1], Alexandros Nanopoulos[2], Lars Schmidt-Thieme[1]

[1] Information Systems and Machine Learning Lab Marienburger Platz 22 University of Hildesheim 31141 Hildesheim Germany
karimi, schmidt-thieme@ismll.uni-hildesheim.de

[2] Department of Business Informatics., Schanz 49, University of Eichstatt-Ingolstadt , 85049 Ingolstadt, Germany
nanopoulos@ku.de

Abstract. A key challenge in recommender systems is how to profile new-users. This problem is called cold-start problem or new-user problem. A well-known solution for this problem is to use active learning techniques and ask new users to rate a few items in order to reveal their preferences. Recently, questionnaire trees (tree structures) have been proposed to build such adaptive questionnaires. In this paper, we improve the questionnaire trees by splitting the nodes of the trees in a finer-grained fashion. Specifically, the nodes are split in a 6-way manner instead of 3-way split. Furthermore, we compare our approach to online updating and show that our method outperforms online updating in order to fold-in the new user into recommendation model. Finally, we develop three simple baselines based on the questionnaire trees and compare them against the state-of-the-art baseline to show that the new-user problem in recommender systems is tough and demands a mature solution.

1 Introduction

Recommender systems help web users to address information overload in a large space of possible options [1]. Collaborative filtering is the traditional technique for recommender systems. Evidently, the performance of collaborative filtering depends on the amount of information that users provide regarding items, most often in the form of ratings. This problem is amplified for new users because they have not provided any rating which impacts negatively on the quality of generated recommendations. A simple and effective way to overcome this 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

 In: T. Seidl, M. Hassani, C. Beecks (Eds.): Proceedings of the LWA 2014 Workshops: KDML, IR, FGWM, Aachen, Germany, 8-10 September 2014, published at http://ceur-ws.org

queries. To address this problem, *active learning* methods have been proposed to acquire the most informative ratings, i.e ratings from users that will help most in determining their interests [3, 4].

Recently, active learning based on tree structures have been proposed by (Golbandi et al. [2]). In [2], the tree structures are ternary because there are three possible answers for queries: "Like", "Dislike", or "Unknown". In datasets like Netflix and MovieLens that the range of the ratings is from 1 to 5, ratings from 1 to 3 are considered as "Dislike" and ratings 4 and 5 are treated as "Like". Moreover, the missing ratings are considered as "Unknown", meaning users do not know the queried item, so they can not rate it.

Nevertheless, [2] was a breakthrough in the literature of active learning for recommender systems. In our previous paper [5], we improved [2] by incorporating matrix factorization into the tree structures and proposing a sampling method to speed up the tree construction algorithm. In this paper, we improve it one step further by upgrading the ternary trees to 6-way trees, meaning the nodes are split in a 6-way fashion. In the 6-way split, there is one child node per each rating from 1 to 5 and one child node for the "Unknown" response. As the 6-way split distinguishes users tastes more precisely, it is expected that the accuracy of the rating prediction also improves. On the other hand, the 6-way split might lead to overfitting, which affects adversely on the accuracy. Therefore, we need a rating prediction model that handles the overfitting issue very well. We apply the 6-way split to two prediction models and show the effect of the overfitting on the accuracy of the 6-way split.

2 Related Work

The idea of using decision trees for the cold-start recommendation was proposed by (Rashid et al. [7]). They tried to formalize the cold-start problem in a supervised learning context and solve it through decision trees. However, they face challenges that force them not to use standard decision tree learning algorithms such as ID3 and C4.5. (Golbandi et al. [2]) improved [7] by advocating a specialized version of decision trees to adapt the preference elicitation process to the new user's responses. As our method relies on [2], we briefly explain it in this section.

Here, each interior node is labeled with an item $i \in I$ and each edge with the user's 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 users response until a leaf node is reached. Here, 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 are not able to rate the queried item because they do not know it. Letting users not to rate the queried items in case they do not know it, is crucial because it happens frequently in recommender systems.

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 the node t:

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

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

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

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

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

To avoid over-fitting, the prediction of the item i is regularized towards its prediction in the parent node r_{si}. λ_1 is the regularization factor. The effect of the regularization for the item i becomes more significant when the number of the ratings in the item profile $\mathcal{D}_i^t$ is less. 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 at 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 the 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 :

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

A naive construction of the tree would be intractable if the number of items and ratings is large. Therefore, (Golbandi et al. [2]) proposes a solution for that. The idea is to expand "Unknown" child nodes in a different way using some statistics collected from "Like" and "Dislike" child nodes.

3 Problem Definition

Let U be a set (of users), I be another set (of items), and $R \subseteq \mathbb{R}$ be a (finite) set of 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 $E \subseteq I \times R$ 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 predictive model $\hat{r} : I \to R$, where the rating of each item can be predicted. For a user profile $E \subseteq I \times R$ let $\hat{R}(E)$ denote the rating predictive model at the leaf one arrives 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 predictive 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

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

The total loss is the loss over all test users:

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

What we call decision tree or questionnaire here really is a multivariate regression tree for instances in $(R^+)^I$ (here called user profiles) with values in $\mathbb{R}^I$. The values in $\mathbb{R}^I$ can be represented by models $\hat{r} : I \to \mathbb{R}$. Other names could be rating prediction tree or recommendation tree.

4 Factorized Decision Trees

In [2], the ratings are predicted based on the item average method, which may seem naive as there are more advanced algorithms, such as Matrix Factorization (MF) [6], which have already shown their superiority over the item average. The reason for using the item average is that building the tree structures is expensive in terms of time. There are many nodes that need to be expanded and per each node there are many candidate items that must be checked. On the other hand, we have to predict ratings in child nodes and compute the error in order to find the best split item. As a result, we need a method for rating prediction that is fast, even though it may not be the best method. Otherwise, building the tree structures would be intractable.

Now the question is "how can we improve rating prediction of the tree structures while keeping its complexity low?" To find a solution for this question, (Karimi et al. [5]) proposed a method, which is called Factorized Decision Trees (FDT). The FDT divides the learning algorithm of the tree structures into two steps. In the first step, the *structure* of the tree structures is learned according to [2]. After constructing the tree, an MF model is trained to learn the *labels* of the tree, in which the labels are the rating predictions in the leaf nodes. In this way, we achieve a learning algorithm that is more accurate and scalable. (Karimi et al. [5]) do not exploit MF during the tree construction algorithm because it causes too much complexity. (Zhou et al. [9]) proposed another approach to incorporate matrix factorization into the tree structures. However, as it has been detailed in [5], it is too complex and is not scalable.

The scalability becomes even more important when we notice how information overload is growing up every day. Until a few years ago, Netflix was the largest data set for recommender systems. But now we have Yahoo Music, containing 717 M ratings, so it is more than 7 times bigger than Netflix. Therefore, we need to think about the scalability of our approaches. Otherwise, even active learning methods are accurate, it is not possible to apply them in big recommender systems.

5 Fine-Grained Questionnaire Trees

First of all, we would like to clarify that trees that have been used for cold-start recommendation are different from decision trees in machine learning. In decision trees, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. But trees in the cold-start problem are not acting as predictive model. They are simply tools that are used to visually and explicitly represent decisions, in which decisions are new users' responses to the queried items. In fact, decision trees in machine learning describe data while in cold-start problem they represent decisions. Therefore, we use *questionnaire trees* term to refer to such trees.

(Golbandi et al.[2]) opts for 3-way splits corresponding to three possible user responses ("Like", "Dislike", and "Unknown"). In datasets like Netflix and

MovieLens that the range of ratings is from 1 to 5, ratings from 1 to 3 are considered as "Dislike" and ratings 4 and 5 are treated as "Like". Moreover, the missing ratings are considered as "Unknown", meaning users do not know the queried item, so they can not rate it. However, it is expected that a more refined split, such as a 6-way split that matches five star levels plus an "unknown" would improve accuracy. The main bottleneck to do so is the overhead caused by increasing the number of nodes. The higher the number of splits, the higher the number of nodes, which requires more time to build questionnaire trees. To make this overhead more clear, we provide an example. Suppose that questionnaire trees are built up to level 2. Given a 3-way split, the total number of nodes is 13. But if the nodes are split based on the 6-way split, questionnaire trees would have 43 nodes. This overhead increases exponentially by increasing the number of queries.

Fortunately, (Karimi et al. [5]) have already proposed a sampling method that drastically speeds up the tree construction algorithm. This method is called Most Popular Sampling (MPS). Instead of checking all candidate items at each node, the MPS checks only those items that are most popular among users associated with the node. Given that MPS is used, the 6-way split can be used to improve the accuracy of rating predictions while the tree learning algorithm is still tractable.

When the new user preference elicitation ends and a couple of ratings are received from the new user, she is treated as a normal user like existing users of the recommender system. On the other hand, there is already a recommendation model for the existing users, which is usually MF. We call it warm MF since it is for users who already have enough ratings in the data set, in contrast to cold (new) users who have a few ratings. Now we need to fill the gap between the new users and the existing users by folding the new user into the warm MF. Specifically, we need to learn the latent features of the new user in the warm MF. A naive approach for doing this is to add the ratings of the new user to the original data set and then retrain the MF with the whole training data set. However, as we have to repeat this process for all new users, it would be very slow. Therefore, we have to switch to online updating. In online updating, using the ratings that the new user has given, only the new user's latent features are updated and the rest of the features including item features and other user features are not touched [8].

Fortunately, the FDT can already provide us with the new user's latent features and there is no need to use online updating. The FDT generates user features for each type of new users, which corresponds to the leaf nodes of decision trees. In this way we learn the new user features with a higher accuracy. Moreover, there is no need for an online updating step to bridge the query prediction model (decision trees) and the recommendation model (MF). In fact, this is a new fold-in approach, in which, given a new user with a few ratings, a subset of training users who have the same ratings like the new user are selected and then the new users features are trained using all ratings of these users. In our experiments, we found out that FDT can become even faster if only user

features are updated and the rest of the features are fixed to the warm MF. However, the accuracy is slightly affected.

6 Experimental set up

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 $\#queries$. The RMSE of user u is computed as follows:

$$RMSE_u = \sqrt{\frac{1}{|D_u^{test}|} \sum_{(i,r) \in D_u^{test}} (r - \hat{r}_{ui})^2} \tag{4}$$

where 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_{ui} is the true (actual) rating. Thus, we examine the problem of selecting at each step, the item for which each new user u will be queried to provide a rating. The item has to be selected in order to minimize the $RMSE$. The RMSE of each test user is measured separately and then the average RMSE over all test users is reported.

We report the performance of 6-way questionnaire trees based on two predictive models: item average (6-way-AVG) and matrix factorization (6-way-FDT). Correspondingly, we choose two baselines: 3-way-AVG [2] and 3-way-FDT [5].

We implemented [2] by ourself in java. First, we followed the same hyper-parameters reported in [2] to calibrate our results against it and make sure that our implementation was correct. Then, in our experiments, we changed one of the hyper-parameters: (Golbandi et al. [2]) do not expand nodes in which the the number of ratings is fewer than $\alpha = 200000$ and stops the learning. The goal is to save runtime. In our experiments, we set α to zero because Most Popular Sampling (MPS) [5] is already able to save runtime and there is no need to stop the learning. The results show that this setting is significantly beneficial. For $\alpha = 200000$, the RMSE is 0.971 after 5 queries but for $\alpha = 0$ the RMSE would be 0.958.

As (Golbandi et al. [2]) conduct their experiment on the Netflix data set, we also run our experiments on this dataset. Since the data set is large, the experiments are done in one fold, the same evaluation protocol as [2]. 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, the training set and the test set, containing 75% and 25% users, respectively. The tree is learned based on the ratings of training users in the training data. The ratings of training users in the original Netflix test split is considered as validation data in our experiments to find the hyper-parameters of MF. The users in the test set are assumed to be new users. The ratings of test users in the

Netflix training dataset are used to generate the user responses in the interview process. To evaluate the performance after each query, the ratings of test users in the Netflix test data are used.

We will also compare our work to three simple baselines. The goal of this comparison is assess the difficulties of the new-user problem. These three baselines are as follows:

- **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, s most popular items are found based on all ratings available in the dataset. Then we start to build 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.

6.1 Results

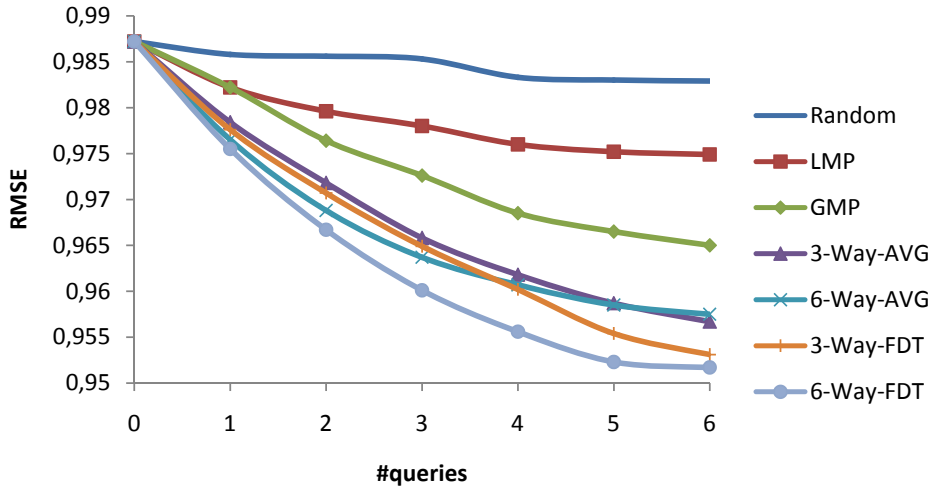

Fig. 1. RMSE results of 6-way split based on FDT (6-way-FDT), 6-way split based on item average (6-way-AVG), 3-way split based on FDT (3-way-FDT), and 3-way split based on item average (3-way-AVG).

Figure 1 shows the results of three simple baselines, 3-way-avg [2], 3-way-FDT [5], 6-way-AVG, and 6-way-FDT. All results are based on MPS where the sampling size is 200. First we discuss about the simple baselines. As the results

show, random item selection performs very badly and gains almost nothing after 8 queries. LMP doesn't work well either. Among the three simple baselines, GMP is the best, although it is still much worse than Bootstrapping. Table 1 shows some statistics which can justify these results. This table shows the probabilities of receiving different responses from new users by each method. The main reason that the random selection does not perform well is that it chooses items that will not be rated by new users. The probability that the 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 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. LMP and GMP receive more ratings compared to the random selection, that is why their performance also improves in Figure 1.

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
Bootstrapping and FDT	0.18	0.15	0.67

Coming back to Figure 1, as we expected the 6-way-AVG beats 3-way-AVG because it provides more refined splits. 6-way-FDT further improves the 6-way-AVG since it leverages MF for rating prediction. The benefit of using MF for rating prediction instead of the item average is more clear in the 6-way split compared to the 3-way split. The reason is that the accuracy we gain at each level is the summation of the improvements of all users at the corresponding level. The higher the number of users, the larger the improvement. A grid search methodology was followed to find hyper-parameters, which are reported in table 2.

After 5 queries, 6-way-AVG converges to 3-way-AVG and even starts to become worse with the sixth query. This happens because, as we go down to the deeper layers of questionnaire 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 accuracy. Although the predictions are still regularized towards the predictions in the parent node, this regularization

might not be enough to compensate for the effect of less training data in such nodes. However, 6-way-FDT does not suffer from this problem because it does not use hierarchical regularization, instead it exploits typical ℓ_2 regularization. Due to the same reason, 3-way-FDT outperforms 6-way-AVG after 4 queries.

Regarding the running time, 6-way-FDT and 3-way-FDT are slower than 6-way-AVG and 3-way-AVG since these method need to train a MF model. In our experiments, training a MF model takes around 3 hours, which considering the gained improvement, it pays off.

Table 2. Hyper-parameters of MF in 6-way-FDT in all levels. α is the learning rate and λ is the regularization factor.

level	α	λ
1	0.0011	0.016
2	0.0015	0.007
3	0.0013	0.005
4	0.0013	0.005
5	0.0013	0.009
6	0.0004	0.03

We finish this section by comparing the FDT to online updating [8]. To use online updating, first decision trees are built as in [2]. Then in the leaf nodes, the user features are retrained only based on the received ratings from the root node to the leaf node. Table 3 reflects the RMSE after each query. Clearly FDT outperforms online updating by a large margin. This happens because FDT uses all ratings of the training users who are similar to the new user to train the user features. However, online updating uses only a few ratings that are received from the new user. The more the number of ratings, the better the accuracy of the learned user features. Interestingly, online updating is even worse than Boot [2], which is based on item average prediction. However, as the Boot method does not provide the latent features, it cannot be used to fold the new user into the warm MF model.

Table 3. The RMSE of online updating in MF and FDT after each query

	0	1	2	3	4	5
Online	1.056	1.0290	1.0056	1.0008	0.9948	0.9914
FDT	0.9872	0.9776	0.9707	0.9649	0.9602	0.9554

7 Conclusion

Using active learning to build adaptive questionnaire trees is the promising approach to address the cold-start problem in recommender systems. The performance of questionnaire trees can be improved by splitting the nodes in a finer-grained fashion, i.e. one child node per each possible rating (including the "Unknown" answer).

As the future work, we plan to use other data sets, in which the maximum rating is higher than 5. For example, in EachMovie, the range of ratings is from one to six, or in IMDb, it is from one to ten. The hypothesis is that opting for the higher number of splits, i.e. 7-way and 11-way splits respectively, may lead to a better accuracy. On the other hand, there might be limitation in the accuracy gained by increasing the number of splits. One needs to verify this hypothesis.

References

1. G. Adomavicius and A. Tuzhilin. 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, 2005.
2. N. Golbandi, Y. Koren, and R. Lempel. Adaptive bootstrapping of recommender systems using decision trees. In *WSDM*, pages 595–604. ACM, 2011.
3. A. S. Harpale and Y. Yang. Personalized active learning for collaborative filtering. In *SIGIR*, pages 91–98. ACM, 2008.
4. R. Jin and L. Si. A bayesian approach toward active learning for collaborative filtering. In *UAI*, 2004.
5. R. Karimi, M. Wistuba, A. Nanopoulos, and L. Schmidt-Thieme. Factorized decision trees for active learning in recommender systems. In *25th IEEE International Conference on Tools With Artificial Intelligence (ICTAI)*, 2013.
6. Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42:30–37, 2009.
7. A. M. Rashid, G. Karypis, and J. Riedl. Learning preferences of new users in recommender systems: an information theoretic approach. *SIGKDD Explor. Newsl.*, 10(2):90–100, Dec. 2008.
8. S. Rendle and L. Schmidt-Thieme. Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In *ACM Conference on Recommender Systems (RecSys)*, pages 251–258. ACM, 2008.
9. K. Zhou, S.-H. Yang, and H. Zha. Functional matrix factorizations for cold-start recommendation. SIGIR '11, pages 315–324. ACM, 2011.