# Factorized Decision Trees for Active Learning in Recommender Systems

Rasoul Karimi, Martin Wistuba, Alexandros Nanopoulos, Lars Schmidt-Thieme
Information Systems and Machine Learning Lab (ISMLL)
Samelsonplatz 1, University of Hildesheim, D-31141 Hildesheim, Germany
Email:[karimi,wistuba ,nanopoulos,schmidt-thieme]@ismll.uni-hildesheim.de

***Abstract*—A key challenge in recommender systems is how to profile new users. A well-known solution for this problem is to use active learning techniques and ask the new user to rate a few items to reveal her preferences. The sequence of queries should not be static, i.e in each step the best query depends on the responses of the new user to the previous queries. Decision trees have been proposed to capture the dynamic aspect of this process. In this paper we improve decision trees in two ways. First, we propose the Most Popular Sampling (MPS) method to increase the speed of the tree construction. In each node, instead of checking all candidate items, only those which are popular among users associated with the node are examined. Second, we develop a new algorithm to build decision trees. It is called Factorized Decision Trees (FDT) and exploits matrix factorization to predict the ratings at nodes of the tree. The experimental results on the Netflix dataset show that both contributions are successful. The MPS increases the speed of the tree construction without harming the accuracy. And FDT improves the accuracy of rating predictions especially in the last queries.**

## I. Introduction

Recommender systems help web users to address information overload in a large space of possible options [1]. 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.

Collaborative filtering is the traditional technique for recommender systems which 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 of the ratings of some other (usually, the $N$ most similar) users for the same item [2]. 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) [3] is a superior prediction model compared to other approaches [3].

Evidently, the performance of collaborative filtering depends on the amount of information that users provide regarding items, most often in the form of ratings. However, a well identified problem is that users are reluctant to provide information for a large amount of items [4], [5]. This fact 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 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 [5], [4].

The query selection process can be personalized or non-personalized. In the non-personalized way, at first a set of items are selected according to some criteria such as popularity or entropy [6] and then they are shown one by one to the new user to rate them. In this way, all new users receive the same set of queries regardless of their responses to the previous queries. Due to this limitation, it does not provide good performance [7].

In the personalized approach, the queries are not selected beforehand but they are selected based on the new user's responses to the previous queries. New users receive different queries depending on how they have rated the previous queries. Although the previous ratings should be taken into account to find the next query, they cannot be found online, i.e when new users are being queried. This is because finding the queries is time consuming and users are not willing to wait for being asked the next query. Therefore, we need to train a model offline and use it when new users come to the system. Decision trees have already been proposed as such models [8], [7].

Initially, new users are at the root of the tree. The first query is asked and depending on how new users rate it, they move down to different child nodes. This process continues until enough queries are asked. At each node of the tree, the new user is linked to the training users who have the same ratings to all queried items so far. The goal of this connection is to leverage their ratings to predict the ratings for the new user. The predicted rating of each item is simply the average over all ratings that the associated users have given to the target item (item average) [7].

In this paper we make the following contributions:

- Formulate active learning for recommender systems. The formulation is then used to compare the proposed method against the baseline.
- Propose the Most Popular Sampling (MPS) method to speed up the tree construction algorithm. In each node, instead of checking all candidate items, only those which are popular among users associated with the node are examined. The results show that it increases

[1] www.netflixprize.com

the speed of the tree construction without harming the accuracy.

- Develop a new algorithm to build decision trees. It is called Factorized Decision Trees (FDT) and exploits matrix factorization to predict the ratings at nodes of the tree. Our results indicate that it improves the accuracy especially in the last queries.

## II. Related Work

In this section, we first summarize the related work on active learning. Next, we focus on the related work that applies active learning in recommender systems.

### A. Related Work on Active Learning

Cohn et al. [9] describe how optimal data selection techniques can be applied to statistically-based learning algorithms like a mixture of Gaussians and locally weighted regression. The algorithm selects instances that minimize the expected error on test data. There are two main limitations for this method: at first, it assumes that the distribution of test data is known which is not always the case. Second, it might choose a query which is optimal for improving the prediction model but the Oracle is not able to label it. Therefore, knowing the answer of this query is not possible. Roy and McCallum [10] eliminated these two issues by considering a limited pool data and then estimating the true test distribution from this pool. However the computation time of this method is high, especially when the size of the pool data is large. Moreover, the estimation of the test data distribution from the pool data might be inaccurate. Baram and et al. [11] proposed a framework based on a multi-armed bandit algorithm to combine several active learning algorithms. The idea behind this framework is that the best active learning algorithm depends on the dataset and should be determined at run time. In this framework all active learning algorithms have chances to select queries.

Osugi and et al. [12] proposed a lighter version of [11] that includes only two active learning algorithms. One algorithm selects examples which are close to decision boundary (exploitation) and the other algorithm selects examples that are far from the decision boundary (exploration). Nguyen and Smeulders [13] offer a framework to incorporate clustering into active learning. This method, which is based on logistic regression, chooses samples close to the classification boundary and samples which are cluster representatives. Furthermore, within the set of cluster representatives, it starts with the highest density clusters first. Poupart [14] proposed an idea to apply Reinforcement Learning (RL) for the active learning problem. This method explicitly models the sequence of queries with a Markov Decision Process (MDP) and learns the best sequence of queries with RL. The introduced idea has been applied for patient treatment in a hospital that is different from pool-based active learning problems. In order to apply this idea to the pool-based active learning problem, one should define state, action, and a reward function according to the characteristics of active learning which is non-trivial.

### B. Related Work on Active Learning for Recommender Systems

Active learning, in the context of the new-user problem, was introduced by Kohrs and Merialdo [15]. This work suggested a method based on nearest-neighbor collaborative filtering, which uses entropy and variance as the loss function to identify the queried items. Al Mamunur et al. [6] expanded this work, by considering the popularity of items and also personalizing the item selection for each individual user. Boutilier et al. [16] applied the metric of expected value of utility to find the most informative item to query, which is to find the item that leads to the most significant change in the highest expected ratings.

Jin and Si [4] developed a new active learning algorithm based on AM which is similar to applying active learning for parameter estimation in Bayesian networks [17]. This method uses the entropy of the model as the loss function. However, this work does 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, this work proposes to use a Bayesian network to take into account the reliability of the current model. This Bayesian approach is, however, complex and intractable for real applications (demands excessive execution time). Harpale and Yang [5] extended [4] 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 for. Karimi et. al [18] 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 in the order of magnitude faster than the Bayesian method.

Karimi et. al [19] developed a non-myopic active learning which capitalizes explicitly on the update procedure of the MF model. Initially, this method queries items that if the new user's features are updated with the provided rating, it will change the features as much as possible. Its goal is to explore the latent space to get closer to the optimal features. Then, it exploits the learned features and slightly adjusts them. Karimi et. al. [20] by being inspired from existing optimal active learning for the regression task, exploits the characteristics of matrix factorization and develops a method which approximates the optimal solution for recommender systems. Karimi et. al. [21] improved the most popular item selection according to the characteristics of MF. It finds similar users to the new user in the latent space and then selects the item which is most popular among the similar users.

The idea of using decision trees for cold-start recommendation was proposed by Al Mamunur et. al [8]. Golbandi et. al [7] improved [8] by advocating a specialized version of decision trees to adapt the preference elicitation process to the new user's responses. Zhou et. al [22] modified [7] by associating 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 believe that [22] is too expensive both in terms of time and memory. The computation complexity for constructing decision trees is $O(q \sum_i N_i^2 + l|I|k^3 + l|I|^2k^2)$ where $q$ is the depth of the tree, $N_i$ is the number of ratings by user $i$, $l$ is the number of nodes, $|I|$ is the number of items, and $k$

is the number of latent features [22]. Imagine we are going to apply this method for the Netflix dataset. In this dataset, $\sum_i N_i^2 \approx 6.48^{10}$, $|I| \approx 18$k, $k$ is usually $> 50$. The number of nodes $l$ is $\sum_{i=0}^{q} 3^i$ , so let's say $q = 7$, then $l$ would be 3280. Considering all these numbers, applying [22] for large datasets is very expensive. The complexity becomes even larger when we note that decision trees are not built only once. After building decision trees in one iteration, item features are updated and again decision trees are build using the updated item features. And the loop continues until the convergence is met. Although [22] conduct their experiments on the Netflix data set, unfortunately the authors do not report the running time of their algorithm. Moreover, in addition to the time complexity, the required memory to store decision trees is also large. We have to store user features of all nodes to update item features after building the tree.

## III. Background

### A. Matrix Factorization

Matrix Factorization (MF) is the task of approximating the true, unobserved ratings-matrix $R$ by $\hat{R} : \mathbb{R}^{|U|\times|I|}$ where $U$ is the set of users and $I$ is the set of items. It maps both users and items to a latent space of dimensionality $k$. In this space, user-item interactions are modeled as inner products. In the latent space, each item $i$ is represented with a vector $h_i \in R^k$. The elements of $h_i$ indicate the importance of factors in rating item $i$ by users. Some factors might have higher effect and vice versa. In the same way, each user $u$ is represented with a vector $w_u \in R^k$ in the latent space. For a given user the element of $w_u$ measure the influence of the factors on user preferences. Different applications of MF differ in the constraints that are sometimes imposed on the factorization. The most common form of MF is finding a low-rank approximation (unconstrained factorization) to a fully observed data matrix minimizing the sum-squared difference to it.

The resulting dot product, $h_i^T w_u$, captures the interaction between user $u$ and item $i$. However, the full rating value is not just explained by this interaction and the user and item bias should also be taken into account. This is because part of the rating values is due to effects associated with either users or items, i.e biases, independent of any interactions.

By considering the user and item bias, the predicted rating is computed as follows [3]:

$$\hat{r}_{ui} = \mu + b_i + b_u + h_i^T w_u \tag{1}$$

where $\mu$ is the global average, $b_i$ is the item bias and $b_u$ is the user bias. The major challenge is computing the mapping of each item and user to factor vectors $h_i, w_u \in R^k$. The mapping is done by minimizing the following squared error [23]:

$$Opt(D,W,H) = \sum_{(u,i)\in D} \Big( (r_{ui} - \mu - b_u - b_i - h_i^T w_u)^2 + \lambda(\|h_i\|^2 + |w_u\|^2) + \gamma(b_i^2 + b_u^2) \Big) \tag{2}$$

where $\lambda$ is the regularization factor, and $D$ is the set of the $(u,i)$ pairs for which $r_{ui}$ is known, i.e the training set $D$. The usual method to train MF is stochastic gradient descent [3]. 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 2. After each epoch, the error of the loss function is checked. If the error is smaller than $\epsilon$, the training stops. Otherwise it continues until $L$ iterations.

### B. Bootstrapping Recommendation by Trees

As our method relies on [7], we briefly explain it in this section. In [7] decision trees are used for bootstrapping a recommender system. In the tree, 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.

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. The predicted rating of item $i$ at the node $t$ is:

$$\hat{r}^t(i) = \frac{sum(t)_i + \lambda_1 \hat{r}^p(i)}{n(t)_i + \lambda_1} \tag{3}$$

where $sum(t)_i$ is the sum of ratings of item $i$ in node $t$, $n(t)_i$ is the number of ratings of item $i$. To avoid over-fitting, predictions at each node are regularized towards the predictions at the parent node. $\lambda_1$ is the regularization factor. The smaller the number of ratings, the larger the regularization.

The squared error associated with node $t$ and item $i$ is: $e^2(t)_i = \sum_{u\in U_t \cap R(i)} (r_{ui} - \hat{r}_{ui})^2$ where $R(i)$ stands for the set of users rating item $i$. Also, the overall squared error at node $t$ is: $e^2(t) = \sum_i e^2(t)_i$ .

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 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 $Err_t(i) = e^2(tL) + e^2(tD) + e^2(tU)$. Among all candidate items, the item which minimizes the following equation is the best :

$$splitter(t) = \underset{i\in I}{\operatorname{argmin}}\ Err_t(i) \tag{4}$$

## IV. Problem Settings

The cold-start problem in recommender systems is usually studied from the perspective of active learning [4], [5], [19], [20], [21]. The reason is because of an analogy between this problem and a similar problem which exists in machine learning community. In supervised machine learning, sometimes there is not enough labeled data to train a model. But there is a set of unlabeled data and it is possible to ask their labels from an oracle. However, querying the labels is costly. Therefore, a few instances from the unlabeled data set are selected for querying but those which are effective to improve the accuracy of the model. This situation is very similar to the new user problem in recommender system. As the new user has not rated any item, it is not possible to train a personalized model for her. But there are many items and we can ask the new user to rate (label) them. However, the new user does not wish to be queried too many times. Therefore, a few items should be selected for querying but those which are effective to improve the new user model. Due to this analogy, techniques which are used to ask new users to rate items are usually called *active learning for recommender systems*.

However, there are a few papers which do not explicitly call their works active learning. For example, [7] prefer to name their method as *bootstrapping*. Although the names are different, principally they have the same goal: to learn new user preferences as much as possible with a few questions. In order to make this analogy more clear, we introduce a formal definition for active learning in recommender systems and then compare it with the bootstrapping.

### A. Active Learning in Recommender Systems

Let $U$ be a set (of users), $I$ be another set (of items), and $Y \subseteq \mathbb{R}$ be a (finite) set of possible ratings, e.g., $Y := \{1, 2, 3, 4, 5\}$. Denote by a triple $(u, i, y) \in U \times I \times Y$ a rating $y$ of user $u$ for item $i$.

For a data set $D \subseteq U \times I \times Y$ denote the set of all users occurring in $D$ by

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

Subsets $E \subseteq I \times Y$ are called user profiles. The profile of user $u$ in D is denoted by

$$D_u := \{(i, y) \in I \times Y \mid (u, i, y) \in D\}$$

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

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

Given

- a data set $D \subseteq U \times I \times Y$,
- a loss $\ell : Y \times \mathbb{R} \to \mathbb{R}$, and
- a maximal number $K \in \mathbb{N}$ of queries,

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

$$\ell(D^{\text{test}}; \hat{Y}) := \frac{1}{|D^{\text{test}}|} \sum_{(u,i,y) \in D^{\text{test}}} \ell(y, \hat{Y}(D_u)(i)) \qquad (5)$$

is minimal.

As $D^{test}$ does not exist during the training phase, active learning techniques do not directly optimize equation 5. Most of them select difficult examples, i.e. examples which the current model is uncertain about their labels. Hopefully, the difficult examples will reduce the test error significantly.

### B. Active Learning or Bootstrapping?

After this formal definition, we can have a better comparison between active learning in recommender systems and bootstrapping. In bootstrapping, the best item is the item which minimizes the *train error* (equation 4). But in active learning, the best item is the item which minimizes the *test error* (equation 5). Even this simplification has also been studied in the literature of active learning but with a minor difference. [10] replaces $D$ with the set of unlabeled data to estimate how candidate examples would reduce the test error. In this paper, we follow the same approach which is taken by [7] and choose items which minimize $D$.

## V. Most Popular Sampling

A naive construction of the tree would be intractable if the number of items and ratings is large. Therefore, [7] 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. In this paper, we propose a sampling method which makes the tree construction algorithm even faster. This method is called Most Popular Sampling (MPS).

In each node, instead of checking all candidate items, only those which are popular among users associated with the node are examined. Formally, let $I' := \{i \in I \mid Rank(i) < M\}$ in which $Rank(i)$ is the popularity ranking of item $i$ and $M$ is the sampling size. Then the equation (4) is changed as follows:

$$splitter(t) = \underset{i \in I'}{\operatorname{argmin}} \ Err_t(i) \qquad (6)$$

To understand why this heuristic works, one needs to return to a challenge that exists in recommender systems. In recommender systems, users usually provide a few ratings. Therefore, many items do not receive ratings from most of the users. If the candidate item is not popular, most of the users go to the "Unknown" child node. In this case, the predictions in the "Unknown" child node would not be significantly different from the predictions in the current node because the associate users of two nodes and consequently the mean ratings are the same. Moreover, as the number of users in "Like" and "Dislike" nodes are not many, the predictions at these nodes are heavily regularized towards the predictions in current node. Therefore, the predictions at these nodes do not significantly improve too. However, splitting nodes with popular items

distributes users more or less uniformly among child nodes. This would lead to new predictions which do not suffer from the mentioned problems.

## VI. Factorized Decision Trees

Factorized Decision Trees (FDT) improves [7] by incorporating MF into decision trees. The motivation is that as MF outperforms item average in ratings prediction for active users, it makes sense to use it for new users as well. To add MF to decision trees, we should take into account how users have been partitioned in the tree.

Initially, all users are at the root. Then they are partitioned into three groups based on their answer to the first selected query. Each child node represents a set of users who have the same answer to the queried item. As each user goes to one of the child nodes at level 1, the unification of the associated users of all nodes gives us a complete set of users. This phenomena is true for all levels and training users have been completely partitioned among all nodes at each level. Due to this fact, we consider one MF model per each level of the tree (Figure 1).

Formally, let $V_l := \{t_1, ..., t_n\}$ be a set of nodes at level $l$. We define a set of pseudo users $U_l = \{u_1, ..., u_n\}$ in which $u_i$ represents node $t_i$. The ratings of each pseudo user $u_i$ is the unification of all ratings of associated users of the corresponding node $t_i$. A MF model is trained with $|U_l|$ users and $|I|$ items to predict $(\hat{y}; E)$. In this way, MF is trained using the complete version of the dataset while the structure of the tree is also kept. Another possibility is to have one model per each node $t_i$. But in this case, we would lose part of the data which is not included in the ratings of the corresponding pseudo user $u_i$. Also, we need to train more MF models which makes the tree construction slower.

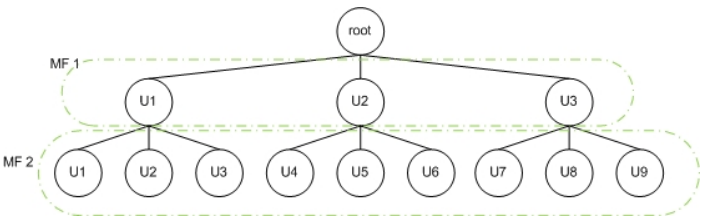


Fig. 1. The first two levels of decision trees. FDT considers one MF model per each level. The number of pseudo users is equal to the number of nodes and the number of items is fixed in all levels.

Algorithm 1 describes the details of FDT. At first, the *structure* of decision trees is learned according to [7] with a minor difference. [7] randomize the item selection process in the way that items that more significantly reduce the error would have more chances to be selected as splitter. The randomization is useful for online evaluations and also for asking new users to rate multiple items in each query. As we do not conduct online evaluation and also we ask new users to provide ratings for one item in each query, the randomization is removed and the splitter item is selected in a fully deterministic manner.

After constructing the tree, a post processing is done on top of the tree to learn the *labels* of the tree, i.e the rating predictions. For each level, the best hyper-parameters of MF are found and then MF is retrained with those parameters. Finally, the predictions are updated using MF. We do not exploit MF during the tree construction because it causes too much complexity. We will return to this issue soon. At level 0, the root of the tree, there is only one node. It means we should train a MF with one pseudo user. However, it is not expected that a MF model with a single user significantly improves the item average. Therefore, for the root node we switch to the item average and compute the predictions similar to [7].

The maximum level of the tree is $K$ which is the maximum number of queries. As the number of pseudo users varies at each level of the tree, the data set of each level is different from other levels. Therefore, we have to do hyper-parameter search per each level. The hyper-parameters of MF includes: number of latent factors $k$, learning rate $\alpha$, user and item feature regularization $\lambda$, and user and item bias regularization $\gamma$.

**Algorithm 1** The algorithm of Factorized Decision Trees

```
Construct decision trees according to [7]
for level l = 1 to K do
    for each α do
        for each λ do
            Train MF
            Compute error on the validation data
        end for
    end for
    Retrain MF with the best hyper-parameters (α, λ)
    Update rating predictions of pseudo users at level l
end for
```

In order to analyze the complexity of FDT, first we need to know the complexity of Bootstrapping [7] because FDT relies on it to build the tree. The complexity of Bootstrapping is $O(d \cdot \sum_{u \in U} |R(u)|^2)$ where $d$ is the depth of the tree and $\sum_{u \in U_t} |R(u)|$ is the total number of ratings of the dataset [7]. According to the algorithm 1, FDT would also have the same complexity, plus the overhead of training MF model at each level of decision trees. In the other hand, the time complexity of MF is $O(|D| \cdot k \cdot L)$ [24]. Hence, the total time complexity of FDT is $O(d \cdot \sum_{u \in S_t} |R(u)|^2 + d \cdot |D| \cdot k \cdot L)$. Please note that the complexity of finding the hyper-parameters is not taken into account like [7]. Compared with the time complexity of functional matrix factorization (fmf) [22], FDT is much less complicated. As it was already pointed out, the complexity of fmf is $O(q \sum_i N_i^2 + l|I|k^3 + l|I|^2k^2)$ which makes it really slow for large datasets like the Netflix.

So far, MF did not have any role on building the tree. The tree is built according to [7] and afterwards MF is used to change the predicted ratings at each node. But this is not optimal. Now that MF is used for rating prediction, the best split item of each node is the one which reduces the error of MF not the item average model. However, choosing the split item of each node based on MF poses challenges which makes it intractable.

Suppose we want to select the $q$-th query. It means that we are in a node at the $(q-1)$-th level of decision trees. For this node, there are $|I| - q - 1$ candidate items. The best item is the one which minimizes the error of MF in the next level. However, to train MF, we need to expand all nodes at level

TABLE I. RMSE RESULTS OF DECISION TREES FOR VARIOUS SAMPLING NUMBER. TIME IS DISPLAYED IN HOURS:MINUTES MEASURED ON A 4 CORES MACHINE EACH CORE 2.4 GHZ

| #samples | time | initial error | query 1 | query 2 | query 3 | query 4 | query 5 | query 6 | query 7 | query 8 | query 9 | query 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 200 | **04:25** | 0.9872 | 0.9784 | 0.9718 | 0.9658 | 0.9618 | 0.9587 | 0.9567 | 0.9553 | 0.9545 | 0.9539 | 0.9536 |
| 1000 | **12:30** | 0.9872 | 0.9784 | 0.9718 | 0.9661 | 0.9620 | 0.9589 | 0.9568 | 0.9555 | 0.9546 | 0.9540 | 0.9537 |
| 17770 | **119:54** | 0.9872 | 0.9784 | 0.9718 | 0.9661 | 0.9620 | 0.9589 | 0.9568 | 0.9555 | 0.9546 | 0.9540 | 0.9537 |

$q-1$ to generate a new dataset including all pseudo users at level $q$. Therefore, the best selected item of each node is not independent of the rest of the node exist at the same level. It means we have to check all selections and choose the one with the minimum error. However, checking all selections would be very expensive. At level $l$, there are $3^l$ nodes, therefore the total number of selections which must be examined is $(|I|-i-1)^{3^l}$. In the other hand, the time complexity of matrix factorization learning algorithm is $O(L\times|D|\times k)$ [24]. Therefore, the complexity of finding the best $q$-th query is $O((|I|-i-1)^{3^{q-1}}\times L\times|D|\times k)$. Considering all levels of the tree, constructing the tree in this way would be intractable.

In general, the number of nodes at level $i$ is $3^i$ and the number of users is equal in all levels. However, there are two situations where this general rule is broken. Suppose we are at level $i$. While we are splitting nodes, we might encounter a node which among its associated users, there is no user who have specific answer to the split item. For example, there is no user who likes the split item. In that case, the node of the missing answer is created but it is null, i.e it has no associated users. When nodes at level $i+1$ are split, the null node is not split because it has no associated user. Thus, the number of nodes at level $i+2$ would be $3^{(i+2)}-3$ because the three child nodes of the null node are missing in level $i+2$.

The second situation happens when a node is split but the summation of errors at child nodes is larger than the error at the current node. According to [7], such nodes are not expanded. It means again the number of nodes in the next level would be three nodes less than the expected number. Moreover, as the associated users of such nodes get stack in the current node, the number of users in the next level would also be less than the number of users in the current level. We call these nodes as *deadlock* nodes.

In our experiments, from level 6 to 8, we observed a few deadlock nodes. In these levels the training data contains less number of ratings compared to the previous levels because the ratings of the associated users of the deadlock nodes are not used in the next level. However, the number of removed ratings is less than 100 which compared to 100M ratings is neglectable and we can say that the total number of ratings in all levels of decision trees are equal.

## VII. EXPERIMENTAL RESULT

In this section, we examine experimentally the performance of the proposed method.

### A. 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 the *number of queries*. The RMSE is computed as follows:

$$RMSE=\sqrt{\frac{1}{N}\sum_{u,i\in D_u^{test}}(r_{ui}-\hat{r}_{ui})^2} \tag{7}$$

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 $RMSE$ based on the MF model. The RMSE of each test user is measured separately and then the average RMSE over all test users is reported.

To regenerate 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 to 1 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, 8 or 10 queries are asked from each new user and we compare it against [7] which is called *Bootstrapping* in the next section. We first implemented [7] by ourself. Then we followed the same hyper-parameters reported in [7] to calibrate our results against it. After ensuring that our implementation is correct, we changed one of the hyper-parameters in our experiments: [7] do not expand nodes in which the the number of ratings is less than $\beta=200000$ and stops the learning. The goal is to save the runtime. In our experiments we set $\beta$ to zero because MPS is already able to save the runtime and there is no need to stop the learning. The results show that this setting is significantly beneficial. For $\beta=200000$, the RMSE is $0.971$ after 5 queries but for $\beta=0$ the RMSE would be $0.958$. Also, for $\beta=200000$ the learning converges after 6 queries but for $\beta=0$ it continues until 8-th query.

As [7] conduct their experiment on the Netflix dataset, we also run our experiments on this dataset. 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 using the ratings of training users in the training data. To find the hyper-parameters of MF, the ratings of training users in the test data is used (validation data). 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.

TABLE II. THE POPULARITY RANKING OF THE SELECTED ITEMS

| #query | average rank |
|---|---|
| 1 | 136 |
| 2 | 56.6 |
| 3 | 172.4 |
| 4 | 495.8 |
| 5 | 324.3 |
| 6 | 544.9 |
| 7 | 760.1 |
| 8 | 520.6 |
| 9 | 1327.8 |
| 10 | 2564.8 |

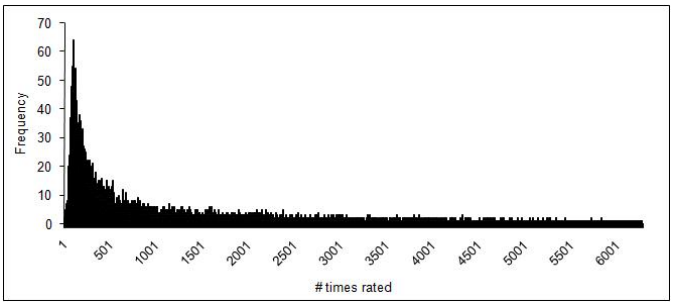


Fig. 2. Distribution of items' rating frequency in the Netflix dataset

### B. Results

First, we report the results of MPS. We do not use MF in this part and the tree is constructed according to [7]. However, to expand nodes only a subset of all candidate items are checked based on MPS. Table I shows how MPS affects the accuracy in different queries. As it is clear, MPS does not harm the accuracy at all while speeds up the tree construction algorithm in the order of magnitude. This observation shows that the best items to query are among popular items and non-popular items are irrelevant. To understand why this happens, we should see the rating frequencies of items. Figure 2 shows the distribution of items' rating frequencies in the Netflix dataset. According to this distribution, most of the items have received less than 1000 ratings. Therefore, splitting users with a non-popular item will move most of the users to the "Unknown" child node and, as it was discussed in section V, recommender systems gain nothing.

However, splitting the nodes with popular items distributes the users of the current among children nodes in a way that there are enough users at children nodes to provide accurate predictions. For example, in the case of the first selected item, 49461 users like it, 29913 users dislike it and 280767 do not rate it. As there are enough users in "Like" and "Dislike" nodes and the number of "Unknown" users are much less than the current node, the new predictions are significantly different from the current node's predictions leading to significant improvement in new predictions.

Table II provides more insights about why sampling popular items is working. In this table, the average popularity ranking of the selected items for each query is shown. In the initial queries, the ranking is rather small. As we go to the lower levels, the ranking increases. In some levels, the average ranking is larger than 200 or 1000. One could expect that for those queries, the sampling would harm the accuracy. However, the results in table I show that this is not the case and we can simply ignore items which their ranking is larger than 200 or 1000. Surprisingly, sampling 200 items even slightly improves the accuracy after 2 queries. To understand the reason of this evidence, the learning algorithm of decision trees should be taken into account. Decision trees are built by choosing the items which minimize the training error. But those items do not necessary minimize the test error as well. Therefore, it makes sense to ignore some items which lead to smaller training error but we are unsure about their test error. The results show that items which are very popular pose lower uncertainty about their test error, thought the reason is not clear for us.

Now we go on to show the results of the other contribution of this paper which is adding MF to decision trees. In this part, MPS was not used and the tree is constructed by checking all candidate items. Table III shows the RMSE of FDT (Factorized Decision Trees) versus the baseline [7]. Both methods start with the same initial error because as it was already pointed FDT uses the item average for rating prediction at the root of the tree. In the first query, Bootstrapping and FDT are more or less equal. It is because there are only 3 pseudo users in the first level. The superiority of MF against the item average is more obvious when MF is trained with many users. As we go down to the lower levels, the number of pseudo users increases and the benefit of rating prediction with MF becomes more clear and after 8 queries, the amount of improvement is 0.005 which is significant on the Netflix dataset. As asking more queries would not significantly improve [7], we stop FDT after 8 queries.

An interesting point is that in the last queries, there is not much improvement in Bootstrapping. It is due to hierarchical regularization used in this method. In the hierarchical regularization, predicted ratings at each node are regularized towards the predictions in their parents. In the nodes which appear at deeper levels, the effect of the regularization increases due to dropping the number of ratings. Although, hierarchical regularization prevents harming the accuracy in going to deeper levels but the accuracy gain would also be small and unjustified. Therefore, we need to come up with a solution which is able to keep improve the accuracy in deeper levels. And this is what FDT does. In FDT, the number of ratings is almost equal in all levels, so we do not need hierarchical regularization. Instead, we exploit typical regularization used in MF, i.e. $L_2$ regularization. As the results show, this approach is quite successful to improve the accuracy at deeper levels. Thus, if new users are willing to answer more queries, FDT is able use them while the performance of Bootstrapping significantly drops in the last queries.

Even small lift in RMSE leads to significant financial benefit for companies [25]. Therefore, the achieved improvements are relevant. The same observation is made in the related works even for smaller datasets [4], [5] as the problem is difficult: there are many candidates items to ask their ratings from new users but new users are willing to rate just a few of them. Moreover, we start without any ratings from the new user which makes the problem more severe.

To find hyper-parameters, a grid search methodology was

TABLE III. RMSE RESULTS OF FACTORIZED DECISION TREES (FDT) VERSUS BOOTSTRAPPING. THE RMSE OF EACH TEST USER IS MEASURED SEPARATELY AND THEN THE AVERAGE RMSE OVER ALL TEST USERS IS REPORTED

| method | initial error | query 1 | query 2 | query 3 | query 4 | query 5 | query 6 | query 7 | query 8 |
|---|---|---|---|---|---|---|---|---|---|
| FDT | 0.9872 | 0.9776 | 0.9707 | 0.9649 | 0.9602 | 0.9554 | 0.9531 | 0.9514 | 0.9498 |
| Bootstrapping | 0.9872 | 0.9784 | 0.9718 | 0.9661 | 0.9620 | 0.9589 | 0.9568 | 0.9555 | 0.9546 |

followed. The hyper-parameters of each level are reported in table IV. In our experiments varying $k$ did not change the results, so we fixed it to 70. Also, the best value of $\gamma$ was 0.0001 for all levels.

TABLE IV. HYPER-PARAMETERS OF MF IN ALL LEVELS

| level | $\alpha$ | $\lambda$ |
|---|---|---|
| 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 |
| 7 | 0.0015 | 0.01 |
| 8 | 0.0006 | 0.02 |

## VIII. CONCLUSION

A main challenge in recommender systems is to provide useful recommendations to new users. Decision trees have already been proposed to build a questionnaire which is used by recommender systems to pose questions to new users. In this way, the system would be able to learn new users' preferences with a few questions and then provide useful recommendations to them. In this paper, we improved decision trees construction in two ways: first, the learning algorithm speeds up by most popular sampling. Second, the accuracy of the rating predictions improves by incorporating MF into decision trees.

As the future work, we plan work on other sampling criteria. Also, we consider to use clustering techniques instead of decision trees to partition users and find an appropriate questionnaire.

## 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*, vol. 17, no. 6, pp. 734–749, 2005.

[2] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl, "GroupLens: Applying collaborative filtering to usenet news," *Communications of the ACM*, vol. 40, no. 3, pp. 77–87, 1997.

[3] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, pp. 30–37, 2009.

[4] R. Jin and L. Si, "A bayesian approach toward active learning for collaborative filtering," in *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, 2004.

[5] A. S. Harpale and Y. Yang, "Personalized active learning for collaborative filtering," in *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2008, pp. 91–98.

[6] A. M. Rashid, I. Albert, D. Cosley, S. K. Lam, S. M. McNee, J. A. Konstan, and J. Riedl, "Getting to know you: Learning new user preferences in recommender systems," in *International Conference on Intelligent User Interfaces (IUI)*. ACM Press, 2002, pp. 127–134.

[7] N. Golbandi, Y. Koren, and R. Lempel, "Adaptive bootstrapping of recommender systems using decision trees." in *WSDM*. ACM, 2011, pp. 595–604.

[8] A. M. Rashid, G. Karypis, and J. Riedl, "Learning preferences of new users in recommender systems: an information theoretic approach," *SIGKDD Explor. Newsl.*, vol. 10, no. 2, pp. 90–100, Dec. 2008.

[9] D. A. Cohn, G. Z., and M. Jordan, "Active learning with statistical models," in *Advances in Neural Information Processing Systems( NIPS)*, 1995.

[10] R. Nicholas and A. McCallum, "Toward optimal active learning through monte carlo estimation of error reduction," in *International Conference on Machine Learning (ICML)*, 2001.

[11] Y. Baram, R. El-Yaniv, K. Luz, and M. Warmuth, "Online choice of active learning algorithms," *Journal of Machine Learning Research*, vol. 5, pp. 255–291, 2004.

[12] T. Osugi, D. Kun, and S. Scott, "Balancing exploration and exploitation: A new algorithm for active machine learning," in *IEEE International Conference on Data Mining (ICDM)*, 2005.

[13] T. Nguyen and A. Smeulders, "Active learning using pre-clustering," in *International Conference on Machine Learning (ICML)*, 2004.

[14] P. Poupart, "Non-myopic active learning: A reinforcement learning approach," in *Google Talk*, March 2009.

[15] A. Kohrs and B. Merialdo, "Improving collaborative filtering for new users by smart object selection," in *International Conference on Media Features (ICMF)*, 2001.

[16] C. Boutilier, R. S. Zemel, and B. Marlin, "Active collaborative filtering," in *Conference on Uncertainty in Artificial Intelligence(UAI)*, 2003.

[17] S. Tong and D. Koller, "Active learning for parameter estimation in bayesian networks," in *Advances in Neural Information Processing Systems( NIPS)*, 2000.

[18] R. Karimi, C. Freudenthaler, A. Nanopoulos, and L. Schmidt-Thieme, "Active learning for aspect model in recommender systems," in *IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*. IEEE, 2011.

[19] R. Karimi, C. Freudenthaler, A. Nanopoulos, and L. Schmidt-Thiemee, "Non-myopic active learning for recommender systems based on matrix factorization," in *IEEE Information Reuse and Integration (IRI)*. IEEE, 2011.

[20] R. Karimi, C. Freudenthaler, A. Nanopoulos, and L. Schmidt-Thieme, "Towards optimal active learning for matrix factorization in recommender systems," in *23th IEEE International Conference on Tools With Artificial Intelligence (ICTAI)*, 2011.

[21] R. Karimi, C. Freudenthaler, A. Nanopoulos, and L. Schmidt-Thiemee, "Exploiting the characteristics of matrix factorization for active learning in recommender systems," in *RecSys*, 2012, pp. 317–320.

[22] K. Zhou, S.-H. Yang, and H. Zha, "Functional matrix factorizations for cold-start recommendation," in *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, ser. SIGIR '11. ACM, 2011, pp. 315–324.

[23] Y. Koren, "Factorization meets the neighborhood: a multifaceted collaborative filtering model," in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '08. ACM, 2008, pp. 426–434.

[24] 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)*. ACM, 2008, pp. 251–258.

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