Learning Multiple-Question Decision Trees for Cold-Start Recommendation

Mingxuan Sun
College of Computing
Georgia Tech.
msun3@gatech.edu

Fuxin Li
College of Computing
Georgia Tech.
fli@cc.gatech.edu

Joonseok Lee
College of Computing
Georgia Tech.
jlee716@gatech.edu

Ke Zhou
College of Computing
Georgia Tech.
kzhou@gatech.edu

Guy Lebanon
College of Computing
Georgia Tech.
lebanon@cc.gatech.edu

Hongyuan Zha
College of Computing
Georgia Tech.
zha@cc.gatech.edu

ABSTRACT

For cold-start recommendation, it is important to rapidly profile new users and generate a good initial set of recommendations through an interview process — users should be queried adaptively in a sequential fashion, and multiple items should be offered for opinion solicitation at each trial. In this work, we propose a novel algorithm that learns to conduct the interview process guided by a decision tree with multiple questions at each split. The splits, represented as sparse weight vectors, are learned through an L_1-constrained optimization framework. The users are directed to child nodes according to the inner product of their responses and the corresponding weight vector. More importantly, to account for the variety of responses coming to a node, a linear regressor is learned within each node using all the previously obtained answers as input to predict item ratings. A user study, preliminary but first in its kind in cold-start recommendation, is conducted to explore the efficient number and format of questions being asked in a recommendation survey to minimize user cognitive efforts. Quantitative experimental validations also show that the proposed algorithm outperforms state-of-the-art approaches in terms of both the prediction accuracy and user cognitive efforts.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information filtering; I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms, Performance, Experimentation

Keywords

Recommender systems, Cold-start problem, Collaborative filtering, Decision tree

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WSDM'13, February 4–8, 2013, Rome, Italy.
Copyright 2013 ACM 978-1-4503-1869-3/13/02 ...$15.00.

1. INTRODUCTION

Recommender systems have been established as a critical tool for many business applications with significant economic impact. Successful systems span a variety of platforms, including Amazon's book recommendations, Netflix's movie recommendations, and Pandora's music recommendations. A popular and effective approach to recommendations is collaborative filtering, which focuses on detecting users with similar preferences and recommending items favored by the like-minded [3, 13, 11, 15]. However, the system would fail to provide recommendations when no preference information is gathered from a new user, which is known as the cold-start problem [6, 17, 22].

Rapid profiling of new users is a key challenge in cold-start recommender systems. One direct way to solicit the preferences of new users is going through an initial interview process [17]. In the interview, the recommender system asks users to provide opinions on selected items and constructs rough user profiles. Asking too few questions may lead to inaccurate estimation of user profiles and the system is unable to provide satisfactory recommendations, while asking too many questions may cause users to abandon the interview. A good interview is targeted at discovering user interests with minimum interactions. Specifically, the process should focus on (1) increasing the recommendation accuracy and (2) minimizing user interaction efforts.

An adaptive interview process is known to improve the accuracy, compared with previous approaches such as methods based on meta-features or a static set of interview questions. Many state-of-the-art work [17, 27, 6] have championed decision trees [4] as a natural fit for adaptive interviews of new users. In a primitive form of the decision tree, each node asks for user opinions on one item and users are directed to subtrees based on whether their answer is *like*, *dislike* or *unknown*. Finally, the average preference of training users within each leaf node is used to generate the recommendation list.

However, most users do not have the luxury to go through the entire item set, thus the system needs to serve a majority of users who have seen only tiny percentage of the items. In this case the constructed decision trees are often extremely unbalanced with the *unknown* branch capturing more than 80% of the users at each split (Figure 1, upper). Many users have to repeatedly select *unknown* multiple times before locating any item they know about. In this case, not only did the system gain little valuable information on users' interests, but the users also easily get bored and may opt out of the system prematurely.

For a better cold-start recommender system, we advocate designs focusing on asking multiple questions at each trial. Displaying multiple questions on one screen increases the chance that a user knows at least one of them and thus allowing for solicitation of more valuable information. This was also suggested by some experimental studies that show users prefer providing opinions on multiple items at a trial [17]. However, within the collaborative filtering context, there has not been much existing work that focuses on building a single decision tree with each node asking multiple questions.

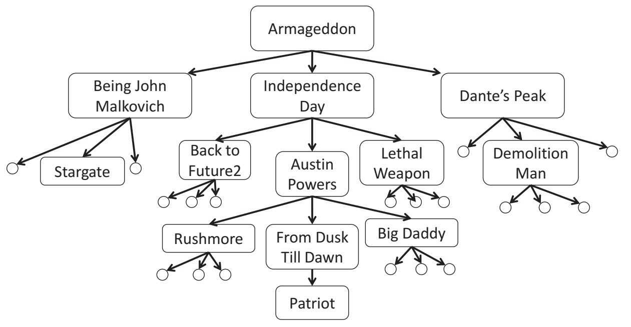
The major contribution of this paper is to develop a tree learning algorithm for cold-start collaborative filtering with each node asking multiple questions. There are two key technical challenges to overcome in learning such a tree structure. First, with multiple questions at each node, it becomes substantially more expensive to search over all possible splits. Instead, we rely on a framework of minimizing expected prediction loss with L_1 regularization to find each split. Second, when asking multiple questions, users with different opinions would be grouped into the same node, and making predictions by averaging training user responses within each node is no longer sufficient. We propose to learn a regressor within each node making use of all the answers from the root to the present node. In our experiments, the algorithm is shown to be able to improve the performance of the cold-start recommendation system. One example of the tree with multiple questions is shown in Figure 1 (lower).

In addition to the prediction accuracy, another essential component for profiling new users is minimizing user efforts in the interaction. Critical features influencing user cognitive efforts include the number of items to ask at each trial, as well as the type of answers collected from the users. User response in common recommender systems can be binary or 5-star rating scale. The 5-star rating response is expected to be more informative than a binary response but it may take more user time. However, few existing work provides a quantitative analysis on the efforts that users would spend on different interfaces.

This issue is addressed in our second contribution which analyzes different strategies in terms of both the prediction accuracy and user efforts. A user study, preliminary but first in its kind, is performed to measure user efforts by the interaction time. Tree-models that ask different number of questions and different type of questions (binary/5-scale) are shown randomly to different users through a web interview interface. The average time spent on each scenario is measured and we are thus able to measure the accuracy against average user interaction time, which reveals the configurations that work best in real-life scenarios. The current quantitative user study that compares $1 - 4$ questions in both binary and 5-star formats is new, to the best of our knowledge.

2. RELATED WORK

There have been a substantial body of literature on collaborative filtering (CF) recommendations, which can largely be classified into memory and model based methods. Memory-based CF methods predict the ratings of items based on the similarity between the test user and training users [20, 3, 10]. Similarity measures include Pearson correlation [20] and Vector cosine similarity [3, 10]. Other memory-based CF methods include item-based CF [21] and a non-parametric probabilistic method based on ranking preference similarities [24]. Model-based CF includes user and item clustering, Bayesian networks [3], and probabilistic latent variable models [15, 26, 11]. The state-of-the-art methods, including the Netflix competition winner, are mostly based on matrix factorization. The factorized matrix can be used to fill out the unobserved entries of the user-rating matrix in a way similar to latent factor analysis [19, 13].



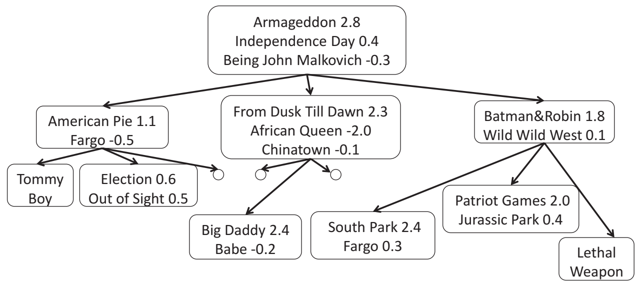


Figure 1: The interview trees learned from the movielens1M dataset. A circle denotes a leaf node with recommendation results. Top: each node asks a single question. The first 5 levels are shown from the 10-level learned tree. Users are branched to left, middle and right subtrees according to the answers *like*, *unknown* and *dislike* respectively. It can be seen that the middle branch of the tree is much longer than the other branches. Bottom: each node asks multiple questions. The first 3 levels out of a total 5 are shown. Users are branched to left, middle and right subtrees according to equation (3) given the answers and corresponding weights.

One difficulty in recommender system is that the performance is affected by factors such as number of users, number of items, and observed total ratings. Unified models have been proposed in [1, 7, 22] by combining both CF methods and content features such as the genre, actor and director of a movie. These methods generally achieve better prediction accuracy on the items that few or no users have ever rated.

Several studies focus on eliciting preference for new users to solve the cold-start problem. It has been demonstrated in [16] that a good elicitation strategy should increase the prediction accuracy with minimum user interaction. Several approaches construct a short interview in which users are asked to provide information on selected items. Earlier attempts [17, 18] construct a static seed set based on the item popularity and informativeness. A later work [5] proposes a greedy algorithm to select the seed set that minimizes the prediction error. The performance of seed-based methods is unsatisfactory because items selected in batch are not adaptive to user responses in the interview process. Active learning methods for collaborative filtering [2, 8, 12, 9] select questions according to criteria such as minimizing the expected distance to the true user model, although these methods are not efficient enough for online user interview.

The IGCN algorithm proposed in [18] uses pre-defined user clusters to adaptively select questions in the interview. An alternative approach using decision trees is also mentioned in [18], where each node is a question and users are directed to subtrees based on their responses. The idea was later developed in [6] as a more disciplined approach that fits the decision tree to user ratings. To improve the prediction accuracy with missing data, the work [27] further integrates the decision tree with matrix factorization to fit the user ratings with a latent factor analysis. The limitation is that only one item is selected to ask at each node of the decision tree. It is highly likely that the user does not know any items in the first several interactions and the estimation would be inaccurate. One alternative suggested in [6] is selecting multiple items to ask at each interaction by bootstrapping multiple trees. The method first generates random trees and then select the best few whose linear combinations fit the training data best. However, this tends to ask more questions compared to building a single tree with multiple questions at each split, because each of the shallow trees would more likely be full and in that case, the number of questions increases linearly with the number of trees.

In this paper, we propose a tree learning algorithm for collaborative filtering with each node asking multiple questions. In contrast to the work [6], our algorithm enables optimization-based estimation procedures for not only the prediction of user profiling but also the selection of questions. Our algorithm achieves significant improvements in terms of the prediction accuracy and user interaction time. Furthermore, it is able to handle heterogeneous preference information including binary and 5-star rating responses.

Existing user study in [17, 18] is measuring the user efforts by the number of pages they views before they rate a minimum number of items for the initial interview. Our user study seeks to discover both the efficient number of questions on each screen, and whether binary or 5-star is more effective in discovering user preferences.

3. DECISION TREE FOR COLD-START

In cold-start recommendations, we assume nothing about new users and an interview process is performed to ask the user a set of questions. For example, in the movie recommendation system, we ask a user questions in the form "Do you like the movie *50 first dates*?" and the user answers *like*, *dislike* and *unknown*, or provides 5-scale ratings, etc. Based on the answers, the system would adaptively ask another set of questions, gradually refine user preference and recommend a set of movies in the end.

In the adaptive interview, we would like to learn a function that maps user responses to user preference profiles from training data. Formally, let r_{ij} denote the observed rating of training user i for item j, where $i = 1, 2, \dots, m$ and $j = 1, 2, \dots, n$. The pairs $(i,\, j)$ are stored in the set $O = \{(i,j) \mid r_{ij} \text{ is observed}\}$. Denote $O_i = \{j \mid r_{ij} \text{ is observed}\}$ the set of items user i rated, $O^j = \{i \mid r_{ij} \text{ is observed}\}$ the set of users that rate item j. Assume there are possible n questions to ask, let x denote the user answer, which is a n-dimensional vector and each entry takes value in the set $\{-1, 0, 1\}$ corresponding to *dislike*, *unknown* and *like*. Given each training user i with the answer x_i, our goal is to learn a user profile $T(x_i)$ that predicts the rating of this user on all the items.

In the simplest form, the user profile $T(x_i)$ is an n dimensional vector with each dimension representing the predicted rating of item j. However, such a representation is computationally challenging and hard to learn in large corpora with thousands or millions of items. A common technique in recommender systems is to obtain lower dimensional models for both users and items by minimizing a prediction loss function between the predicted ratings and true ratings of users. That is,

$$\min_{T,V} \sum_{i} \sum_{j \in O_i} (r_{ij} - T(x_i)^\top v_j)^2, \tag{1}$$

where $T(x_i)$ is a k-dimensional vector, $V = [v_1, v_2, \dots, v_n]$ is a $k \times n$ matrix, and v_j is a k-dimensional vector representing the profile for item j. Usually, T and V are optimized using matrix factorization methods, such as PCA, NMF or nuclear-norm based matrix completion.

A decision tree structure is a natural fit for adaptive interviews where each node corresponds to one or multiple questions to ask. According to the evaluation of a user's answers, the user will be directed to subtrees. Once the user reaches the leaf node, the recommendation list is provided using the user preference learned from training data. Thus, the mapping function $T(x)$ should be viewed as a function induced from the decision tree, which is constructed from training user ratings in a top-down approach. For each node N in the tree, we learn a distinct function $T_N(x)$ for that node. During test time, a user is first assigned to a node based on his/her answers to the interview, then the function in that node is used to obtain his/her user profile.

We adopt the framework in (1) for V and will discuss the details for obtaining V in Section 3.4. However, our main contribution lies in determining the decision tree split with multiple questions, as well as learning the user profile function in each node. Therefore we discuss these first in Section 3.1 - 3.3.

3.1 Multiple Question Decision Tree Construction

In this subsection, we propose our approach of learning to ask multiple questions at each decision tree split. For this goal, there are two challenges to overcome. First, the number of possible questions can be quite large (e.g., we may have $n \sim 10^5$ in movie recommendation system) and searching over all possible splits becomes a combinatorial problem with a prohibitive $\binom{n}{l}$ possibilities to evaluate where l is the number of questions at each node. We solve this problem by relaxing it as an L_1 regularized optimization. Second, since we are keeping the structure of the tree with 3 branches, it is possible that users with different opinions would enter the same node. Therefore different from other decision tree approaches (e.g. [27]), a non-constant function T needs to be learned separately for each node. We solve this second problem by training a linear regressor within each node using all the previously obtained answers as input.

For splitting with multiple items, the idea is to optimize for a sparse weight vector on each item that has only a few non-zeros. Formally, let w denote the weight of items, which is a n-dimensional vector. Let l denote the maximum number of questions at each split. Training users at the current node are split into 3 child nodes L, D and U according to the linear combination of the answers $x_i^\top w$.

We use a modified logistic probability model to determine the split. Let p_i, q_i denote the probability that user i belongs to the L, and D branch respectively:

$$p_i = \frac{1}{1 + c\exp(-x_i^\top w)},$$
$$q_i = \frac{1}{1 + c\exp(x_i^\top w)}. \tag{2}$$

The three subgroups are defined as:

$$L(w) = \{i | p_i > q_i,\ p_i > 1 - p_i - q_i\},$$
$$D(w) = \{i | q_i > p_i,\ q_i > 1 - p_i - q_i\},$$
$$U(w) = \{i | 1 - p_i - q_i \geq \max(p_i, q_i)\}, \tag{3}$$

where c is a parameter controlling the likelihood of falling into different groups. In practice, we find that $c = 2$ works best. In such a case, a user belongs to the L group when $x_i^\top w$ is positive, the D group when $x_i^\top w$ is negative, and the middle group U only when the user answers none of the questions, as shown in Figure 2.

Ideally, we need to optimize such an objective function

$$\begin{aligned}\min_{w,T_L,T_D,T_U} & \sum_{i\in L(w)} \sum_{j\in O_i} (r_{ij} - T_L(x_i)^\top v_j)^2 \\ & + \sum_{i\in D(w)} \sum_{j\in O_i} (r_{ij} - T_D(x_i)^\top v_j)^2 \\ & + \sum_{i\in U(w)} \sum_{j\in O_i} (r_{ij} - T_U(x_i)^\top v_j)^2 \\ \text{s.t. } & \|w\|_0 \leq l,\end{aligned} \tag{4}$$

where T_L, T_D and T_U are the profile functions for the nodes L, D, U, and $\|w\|_0$ denotes the number of non-zeros in w. The objective function first divides the training users into three child nodes, and then computes the squared error within each child node. The goal is to minimize the sum of errors in all the three child nodes. In addition, the constraint $\|w\|_0 \leq l$ determined that w cannot have more than l non-zeros.

We would adopt an alternating minimization strategy that optimizes w and T_L, T_D, T_U iteratively. However, the optimization of w is a complicated non-convex combinatorial problem, therefore we make relaxations to solve it with continuous optimization. In the next two subsections we discuss separately the optimization of w and the computation of T_L, T_D, and T_U.

3.2 Optimization Relaxations

In order to update the weight vector w in (4) with continuous optimization, we adopt two relaxations. The first is to relax the hard partitioning L, D, and U into a soft partitioning, with the probability of x belonging to the subgroups L, D and U to be p_i, q_i, and $1 - p_i - q_i$, respectively. This makes the main objective function smooth in w.

The second relaxation is to append a penalty term on $\|w\|_1$ to the objective function, instead of a hard constraint on the number of nonzeros $\|w\|_0$. This L_1 relaxation approach has been popular in machine learning and signal processing in recent years. The L_1 term is convex and there exist efficient methods to optimize a smooth objective function with such a penalty term [23].

Let $\bar{m}$ denote the number of users reaching the current node, our relaxation problem computes w which minimizes the weighted prediction loss:

$$\begin{aligned}\min_w & \sum_{i=1}^{\bar{m}} p_i \sum_{j\in O_i} (r_{ij} - T_L(x_i)^\top v_j)^2 \\ & + \sum_{i=1}^{\bar{m}} q_i \sum_{j\in O_i} (r_{ij} - T_D(x_i)^\top v_j)^2 \\ & + \sum_{i=1}^{\bar{m}} (1 - p_i - q_i) \sum_{j\in O_i} (r_{ij} - T_U(x_i)^\top v_j)^2 + \lambda\|w\|_1.\end{aligned} \tag{5}$$

To solve (5), we adopt a projected scaled sub-gradient optimization [23]. This algorithm computes the Hessian only for the nonzero part of the current w, and adopts linear-time Barzilai-Borwein subgradient steps for the rest. It is suitable for our task because of its fast convergence rate thanks to the incorporated second-order information, and each iteration is a linear-time operation. The only cubic-time computation is to compute the Hessian inverse on the nonzeros. Since we have usually less than 5 nonzeros, this step would normally take constant time.

By changing the parameter λ, we can find solutions with different numbers of nonzeros in w. In practice, we first binary search between 0 and λ_{max} to find λ_0 such that the number of nonzero entries of w is between 1 and l. Then we search around λ_0 with a finer step size, locating l different solutions with $1, 2, \ldots, l$ nonzeros respectively. From this solution set, we select the one that minimize (5) without the penalty term.

The optimization for w is still non-convex, therefore a good choice of starting point is important. We choose the starting point as the best 1-item question found by the approach in [27] and set $w_s = 1$ and $w_j = 0$ $(j = 1, 2, \ldots, n, j \neq s)$ as our initial value. We find this scheme to work well in practice.

3.3 Computing User Profile Functions

We adopt a linear regression model in each node in order to map different user answers to preferences. The input to the linear model would be all the previous answers that the user has submitted and the output is the user profile. The nodes deeper down the tree would have more information since more answers have been submitted by the user to arrive at the node.

Formally, let t $(t < n)$ denote the number of asked items from the root till the current node, $\bar{x}_i$ denote the answer of user i, which is a $t+1$ dimensional vector including a constant dimension $\bar{x}_{i0} = 1$. Our linear model is $T_L(x_i) = Z_L\bar{x}_i$, $T_D(x_i) = Z_D\bar{x}_i$ and $T_U(x_i) = Z_U\bar{x}_i$, where Z_L, Z_D and Z_U are $(t+1) \times k$ matrices for each individual node in the next level.

We try to best approximate all the observed scores r_{ij} within the node, using all the obtained answers $\bar{x}_i$ and the current item profiles v_j. This gives the following optimization problem:

$$Z_L = \arg\min_Z \sum_{i\in L(w)} \sum_{j\in O_i} (r_{ij} - (Z^\top \bar{x}_i)^\top v_j)^2,$$
$$Z_D = \arg\min_Z \sum_{i\in D(w)} \sum_{j\in O_i} (r_{ij} - (Z^\top \bar{x}_i)^\top v_j)^2,$$
$$Z_U = \arg\min_Z \sum_{i\in U(w)} \sum_{j\in O_i} (r_{ij} - (Z^\top \bar{x}_i)^\top v_j)^2. \quad (6)$$

It turns out that each has a closed form solution:

$$z_L = \left(\sum_{i\in L(w)} \sum_{j\in O_i} (\bar{x}_i \bar{x}_i^\top) \otimes (v_j v_j^\top) \right)^{-1} \left(\sum_{i\in L(w)} \sum_{j\in O_i} r_{ij} (\bar{x}_i \otimes v_j) \right),$$
$$z_D = \left(\sum_{i\in D(w)} \sum_{j\in O_i} (\bar{x}_i \bar{x}_i^\top) \otimes (v_j v_j^\top) \right)^{-1} \left(\sum_{i\in D(w)} \sum_{j\in O_i} r_{ij} (\bar{x}_i \otimes v_j) \right),$$
$$z_U = \left(\sum_{i\in U(w)} \sum_{j\in O_i} (\bar{x}_i \bar{x}_i^\top) \otimes (v_j v_j^\top) \right)^{-1} \left(\sum_{i\in U(w)} \sum_{j\in O_i} r_{ij} (\bar{x}_i \otimes v_j) \right), \quad (7)$$

where z_L, z_D and z_U are large column vectors formed by concatenating the column vectors of the matrices Z_L, Z_D, and Z_U respectively, and $\otimes$ denotes the matrix Kronecker product. The regression inside each node allows different answers to be mapped to different ratings, thus solving the problem of contradicting opinions for different users within a node. At each node, we alternatively optimize (5) and (6) until convergence.

When the amount of training data becomes small along the path reaching the leaf node, the estimation of user profiles may overfit. Following previous papers [27], we apply hierarchical regularization at the current node so that the coefficients of profile Z_L, Z_D and Z_U are shrinking towards the ones at its parent node. For example, the solution with regularization for Z_L is:

$$Z_L = \arg\min_Z \sum_{i\in L(w)} \sum_{j\in O_i} (r_{ij} - (Z^\top \bar{x}_i)^\top v_j)^2 + \lambda_z ||Z - Z_0||^2, \quad (8)$$

where Z_0 is the estimation at the parent node padded with zeros to reach the size of Z at the child node.

3.4 Item Profile Construction

The item profile is initialized using a nonlinear matrix factorization method based on Gaussian process latent variable models [14]. Given an initialized profiles v_j, we can construct a tree T using the algorithm in section 3.1. Given the decision tree T, the user profile estimated at leaf nodes of the tree is $T(x_i) = (Z_k^\top x_i)$, where x_i is the user responses along the path. We then update item profiles v_j $(j = 1, 2, \ldots, n)$ with regularized least square regression:

$$v_j = \arg\min_{v_j} \sum_{i\in O^j} (r_{ij} - v_j^\top T(x_i))^2 + \lambda_v ||v_j||^2. \quad (9)$$

A closed form solution for v_j exists as shown in [27].

3.5 Computational Complexity

The full algorithm is summarized in Algorithms 1 and 2. The computational complexity for updating item profiles v_j for all $j = 1, 2, \ldots, n$ is $O(n|O^j|_{max}k^2 + nk^3)$, where n is the number of items, $|O^j|$ is the number of users who rated item j, and k is the dimension of latent space. For the tree construction, at each node, the complexity for running the split algorithm in [27] is $O(\sum_{i\in users} |O_i|^2 + nk^3)$, the complexity for optimizing w and Z by (5) is $O(\alpha \sum_{i\in users} |O_i| n + \alpha t^3 k^3)$, where $|O_i|$ is the number of observed ratings of user i at the node, α is the iterations, and t is the maximum number of questions asked in the path. The complexity for building the whole tree is thus $O(d \sum_{i=1}^m |O_i| n + \beta n k^3 + \beta t^3 k^3)$, where d is the depth of the tree and β is the number of nodes in the tree. In practice, the tree depth d is no more than 6, the maximum number of questions t is no more than 20, and k is usually selected from 10 to 20.

Algorithm 1 Optimization for Tree Model and Item Profiles

```
Require: The training data $R = r_{ij}|(i,j) \in O$.
Ensure: Estimated decision tree $T$ and item profile $v_j$ $(j = 1, 2, \ldots, n)$.
1: Initialize $v_j$ $(j = 1, 2, \ldots, n)$ using [14].
2: while not converge do
3:     Fit a decision tree $T$ using Algorithm 2.
4:     Update $v_j$ using Equation (9).
5: end while
6: return $T$ and $v_j$ $(j = 1, 2, \ldots, n)$.
```

Algorithm 2 Optimization for Tree Model

```
1: function FitTree(UsersAtNode)
2: Find the best single item $s$ using [27].
3: Initialize $w$ with all zeros but $w_s = 1$.
4: while not converge do
5:     Compute $Z_L$, $Z_D$, and $Z_U$ using Equation (7).
6:     Update $w$ using Equation (5).
7: end while
8: Split users into three groups $L(w)$, $D(w)$ and $U(w)$.
9: if square error reduces after split and depth < maxDepth then
10:     call FitTree($L(w)$), FitTree($D(w)$) and FitTree($U(w)$) to construct subtrees.
11: end if
12: return $T$ with $T(x) = (Z_k^\top x)^\top V$, if $x$ falls in the $k$-th node in the decision tree.
13: end function
```

4. EXPERIMENTS

We examine the estimation framework on three movie recommendation datasets: Movielens1M, EachMovie and Netflix. The details of each dataset are shown in Table 1. The Netflix data we used is a random sample containing about half of the original movies and a quarter of the original users. The prediction performance is evaluated on the test set S with the root mean square error (RMSE), which is defined as:

$$RMSE = \sqrt{\frac{1}{|S|} \sum_{(i,j)\in S} (\hat{r}_{i,j} - r_{i,j})^2}, \quad (10)$$

where $\hat{r}_{i,j}$ and $r_{i,j}$ are the predicted and ground truth ratings for user i and item j, respectively.

We seek to answer three questions in the following:

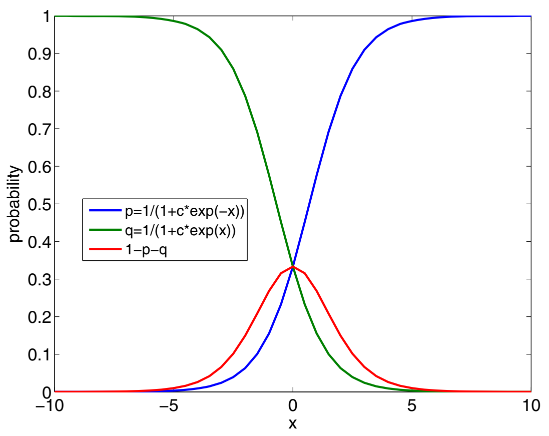


Figure 2: Probabilities of being in the three groups with $c = 2$.

1. For new users, how does the proposed algorithm perform in terms of the prediction accuracy comparing to baselines? How does the performance improve with respect to the number of screens and questions?
2. How many questions do we ask on each screen in order to minimize user interaction time?
3. What types of user responses are more informative and less time-consuming?

In the end, we summarize and analyze insights on how many items to query on each screen, what types of answers users can employ, and how prediction accuracy changes with different settings.

Table 1: Dataset Description

Dataset	Users	Items	Ratings	Scale
MovieLens1M	6040	3952	$1,000,209$	$1-5$
EachMovie	$72,916$	1628	$2,811,983$	$1-6$
Netflix	$120,000$	8000	$11,670,175$	$1-5$

4.1 Cold-Start Prediction Accuracy

In this experiment, we randomly split the data into a training and a test set, each containing 75% and 25% users, respectively. The parameters λ_z and λ_v in the algorithm are selected by 4-fold cross validation on the training set. The dimensionality for the profile, k is fixed to be 15. All the ratings in the training set are used to construct the decision tree. The test set is further split into two disjoint sets: answer and evaluation sets, which contain 75% and 25% rated items for each test user. The answer set is used to simulate the responses of test users in the interview process. The evaluation set is used to evaluate the prediction accuracy after the interview process. The question is in the form "Do you like item j?". For example in the movie recommendation, the question is "Do you like movie *50 first date*?" For binary answer type, where a user is expected to answer *like*, *dislike* and *unknown*, we follow the standard settings [18, 6, 27] and simulate test user responses as the following:

$$x_{ij} = \begin{cases} 1 & (i,j) \in O \text{ and } r_{ij} > 3 \\ -1 & (i,j) \in O \text{ and } r_{ij} <= 3 \\ 0 & (i,j) \notin O. \end{cases} \tag{11}$$

For the EachMovie dataset where the rating scale is $1-6$, we use $r_{ij} > 4$ instead.

Figure 3 compares the performance of our model with several standard baselines. One is the single-question decision tree with matrix factorization, denoted as $w1_{Base}$ [27]. This model has been shown in [27] as the strongest tree model with single-question splits. The other is bootstrapping tree model $w4_{Base}$ [6] which generates random decision trees, and select k-best ones to form a linear combination. In our experiments, the tree generation method in [6] always performs worse than the one in [27]. Therefore in $w4_{Base}$ we adopted the bootstrapping methodology in [6] with the tree generation method from [27], in order for a fairer comparison. We used the best parameters as reported in [27] and [6].

It has been shown that $1 \le k \le 5$ is practical for online interview. We set $k = 4$. Our models $w2$, $w3$, and $w4$ are the trees with maximum number of questions at each node being 2, 3, and 4 respectively.

For all methods, the prediction error measured in RMSE decreases as more questions have been asked (Figure 3). In the first row of Figure 3, our models $w2, w3$ and $w4$ have a big advantage over $w1_{Base}$ when the error is measured per screen, since in each screen we have definitely solicited more information than $w1_{Base}$. The model $w4_{Base}$ also performs better than $w1_{Base}$ in this sense. But more notably, our models $w3$ and $w4$ outperform $w4_{Base}$, which shows the advantage of our multiple-question tree model versus a linear combination of multiple bootstrapped trees.

In the second row in Figure 3 where we compare performance versus number of questions, we see no major difference between $w1_{Base}$ and $w2, w3, w4$ when relatively few questions have been asked. However, our multiple-question tree model is able to ask more questions and further reduce the prediction error, whereas in $w1_{Base}$, the performance cannot improve with trees more than $6-7$ layers, because at this time the number of training users in each leaf node is already very small, essentially making further splits very susceptible to overfitting. The model $w4_{Base}$ in this row showed its inefficiency in needing to ask more questions to obtain the same performance, which comes from the redundancy and inefficient use of information from the multiple trees.

In the third row, we compare performance versus the expected time that users spend in the interview process measured from our user study, described in the next subsection. In this case, $w1_{Base}$ is comparable with our models when relatively few questions have been asked. However, our improvement is more significant if the user is going to spend more than 20 seconds in the recommender system. In that case the performance of single-question decision trees starts to saturate, but our models continue to provide more and more accurate recommendations to the user.

Table 2 shows an example of a user's responses and the recommendation list.

4.2 User Study

In this user study, we measure the expected time that users spend on the interview process through an online interview. Once a user logs in, the interview script loads the precomputed tree model and displays questions of the node on one screen. It proceeds to another screen after the user submits answers. The interview process ends when the user

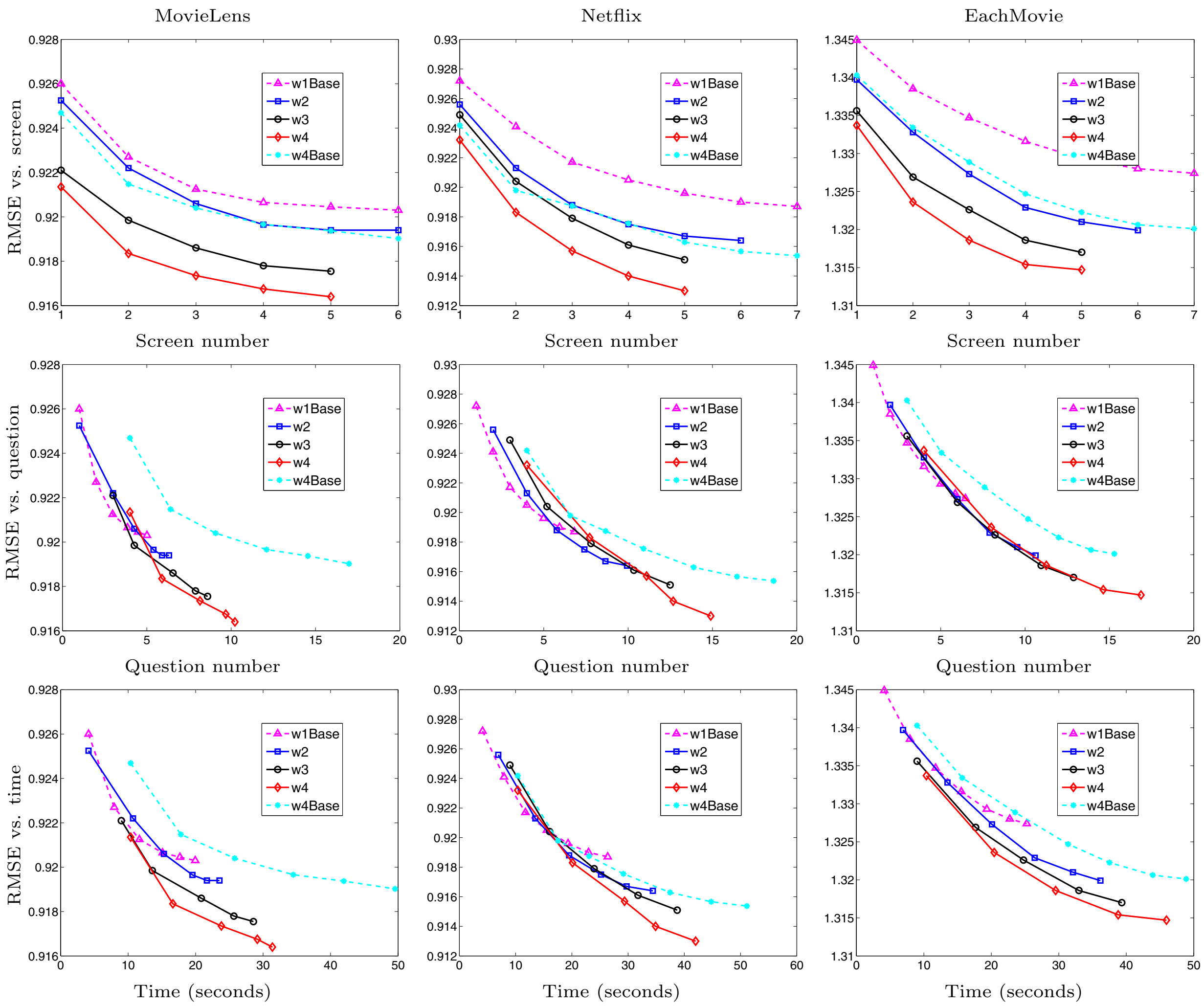


Figure 3: The prediction RMSE with respect to screen number, question number and user time on three datasets. We compare our methods asking 2, 3 and 4 questions with baseline $w1_{Base}$ asking single question and baseline $w4_{Base}$ asking 4 questions. For all methods, the prediction error measured in RMSE decreases as the screen number, question number increases. The first and second row shows that methods asking $2, 3, 4$ questions on each screen performs better than asking one question $w1_{Base}$. The time in the third row is the expected time users spend in the interview process measured from our user study.

reaches a leaf node of the tree model. The trees are always downloaded fully into the local computer in order to save network communication time. There are 8 precomputed tree models in total. For tree models labeled as $w1_{Binary}$, $w2_{Binary}$, $w3_{Binary}$ and $w4_{Binary}$, users are asked 1, 2, 3, and 4 questions maximum at each node respectively and provides binary answers *dislike*, *like*, or *unknown*. For another 4 tree models $w1_{Rate}$, $w2_{Rate}$, $w3_{Rate}$, and $w4_{Rate}$, user responses are in $1-5$ rating scale. A random tree type will be selected when a user logs in. The users are instructed to answer the questions at their first sight, in order to avoid the situation that users linger on a particular page too long, or even search the internet to find particular answers.

In total we are able to collect responses from 76 users. Figure 4 shows our interview web interface. Table 3 compares the expected time to answer k questions on one screen. It reveals that the time increment for users to answer more questions per screen is usually sublinear, with answering 2 questions around 1.5 times slower than asking 1 question, and answering 4 questions around 2.5 times slower. Answering rating scale questions usually adds $2.5-3.5$ seconds to all the 4 settings ($1-4$ questions). The time difference between a rating scale answer and a binary answer seems not to have strong dependency with the number of questions. We suspect this phenomenon still comes from the sparsity of the data: with 4 questions per page, usually the user would not have seen all movies on a page therefore he/she only has to provide ratings for $1-2$ of them.

Figure 5 compares the time of the interview process by asking different number of questions per screen. It can be seen that $w1_{Rate}$ does not seem to be a much better option than $w1_{Binary}$ because the time increases significantly without too many questions being asked. However, the fraction of time increased from $w4_{Binary}$ to $w4_{Rate}$ seems not too big. Although $w4_{Rate}$ costs the longest time (over 50 seconds for 4 screens), it is able to obtain about 14 rated

Table 2: A user's responses to the first two screens of the interview process each with 4 questions, followed by the recommendation list.

(a) Interview Questions

Screen	Questions	Responses
1	The Royal Tenenbaums	Unknown
	Lost in Translation	Like
	Independence Day	Unknown
	Being John Malkovich	Unknown
2	How to Lose a Guy in 10 Days	Like
	Miss Congeniality	Like
	Pulp Fiction	DisLike
	Taxi Driver	Unknown

(b) Top-5 Recommendations

Rank	Movie Title
1	Indiana Jones and the Last Crusade
2	The Sixth Sense
3	The Green Mile
4	Life Is Beautiful
5	Toy Story

Table 3: A comparison of average time to answer n questions on one screen. When the number of questions per screen increases, the increase in the response time is sublinear.

Binary User Response

Ask l questions	$l = 1$	$l = 2$	$l = 3$	$l = 4$
Time (seconds)	4.4097	6.7267	7.8841	10.4232

Rating Scale User Response

Ask l questions	$l = 1$	$l = 2$	$l = 3$	$l = 4$
Time (seconds)	6.9339	9.5906	10.1404	13.8569

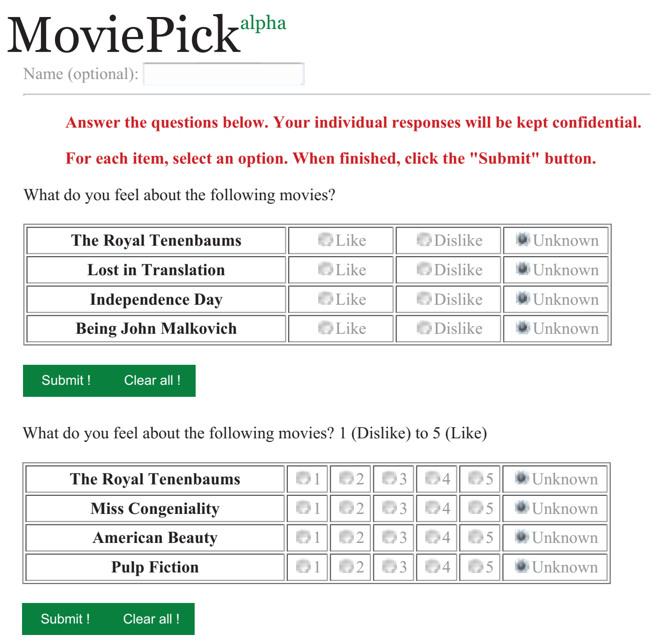
responses, comparing to less than 5 rated responses from $w1_{Rate}$ in 35 seconds.

We plot in Figure 6 the prediction performance for these different settings. The performance differs with data, in MovieLens, the setting $w4_{Binary}$ with asking 4 questions for binary answers performs the best. In the Netflix data, the main advantage of a rated response is that it can achieve superior recommendation performance, with longer surveys. However, time-wise it has only a very small advantage if the total time spent on the survey is less than 30 seconds. In the EachMovie data however, a rated response is always significantly better than a binary one in all accounts. This study can provide material to support decisions in deploying cold-start recommender systems: if the goal is a short survey, then using either binary or rated responses should have no major difference, while rated responses could have a slight advantage. However if the goal is a longer survey for more accurate user responses, it is more likely that seeking rated responses from the user would perform better.

5. CONCLUSIONS

In this paper we propose a new algorithm to learn decision trees with multiple dimensions selected for each split, for applications in cold-start recommender systems. Based on L_1 regularization, a relaxation formulation is proposed to optimize for each split. The new tree has users with heterogeneous answers in the same node. Therefore regressors are learned within each node based on previously given answers. Experiments show that the algorithm outperforms state-of-the-art approaches in terms of both the prediction accuracy and user cognitive efforts. A user study is conducted to understand the efficient number and format of questions being asked in a movie recommendation survey. The conclusion is that the multiple movies per screen approach is more favorable to single movie per screen approach, and asking for a rated response is more beneficial than a binary response, especially in longer surveys.

The current framework focuses on collaborative filtering recommendation and it does not take into account the content features such as user information and item attributes. In the future, we may learn to ask questions based on hybrid information of varying degree of granularity (e.g., genre, subgenre, director, actor, user age). Moreover, there are other types of strategies to explore in terms of minimizing user efforts. One example suggested in [25] supplies a list of items and asks the users to choose some from them. Another similar cold-start problem is to learn the profile for new items by asking a few experienced users to rate them. In this case, asking multiple users at each trial would also be preferable because some users may not respond to a particular question. We suspect that there are some duality between this cold-start item problem and the current one, and the current framework can be extended to handle this new task.



MoviePick$^{\text{alpha}}$

Name (optional):

Answer the questions below. Your individual responses will be kept confidential.

For each item, select an option. When finished, click the "Submit" button.

What do you feel about the following movies?

The Royal Tenenbaums	Like	Dislike	[✓ Unknown]
Lost in Translation	Like	Dislike	[✓ Unknown]
Independence Day	Like	Dislike	[✓ Unknown]
Being John Malkovich	Like	Dislike	[✓ Unknown]

What do you feel about the following movies? 1 (Dislike) to 5 (Like)

The Royal Tenenbaums	1	2	3	4	5	[✓ Unknown]
Miss Congeniality	1	2	3	4	5	[✓ Unknown]
American Beauty	1	2	3	4	5	[✓ Unknown]
Pulp Fiction	1	2	3	4	5	[✓ Unknown]

Figure 4: Interview web interface with 2 types of user input. Upper: binary. Bottom: rating scale.

6. ACKNOWLEDGMENTS

Part of the work was supported by NSF IIS-1116886 and a Yahoo! Faculty Research and Engagement award.

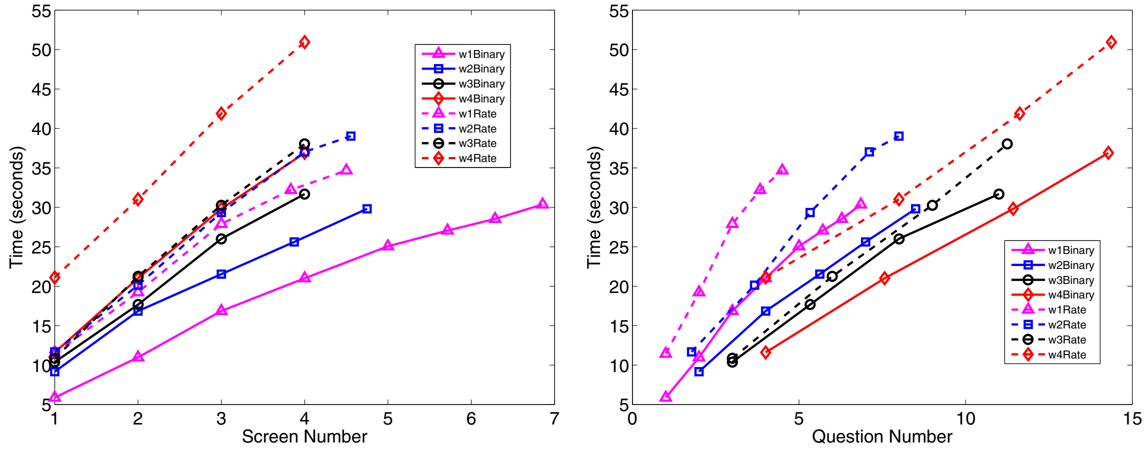


Figure 5: Time of the interview process with different question number per screen. The horizontal axis of left figure is the number of screen and the horizontal axis of right figure is the averaged number of accumulated questions.

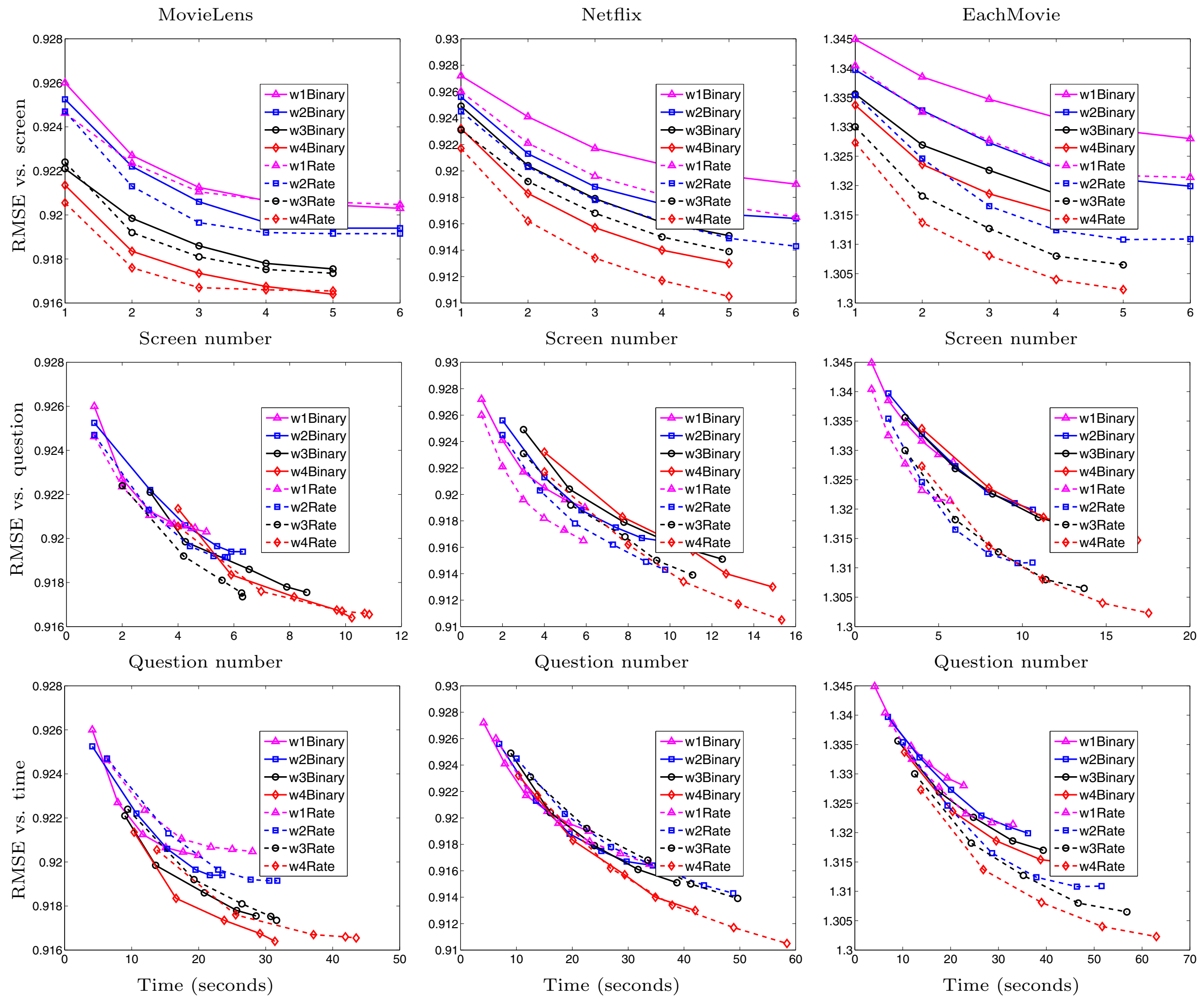


Figure 6: The prediction RMSE with respect to screen number, question number and user time on three datasets given different types of user answers. The *Binary* type expects users to answer *like* and *dislike*. $w2_{Binary}$, $w3_{Binary}$ and $w4_{Binary}$ are the tree models with 2, 3 and 4 questions on each screen respectively. In the *Rate* type, user chooses from a 5-star rating scale. $w2_{Rate}$, $w3_{Rate}$ and $w4_{Rate}$ are the tree models with 2, 3 and 4 questions on each screen respectively. Generally, rating scale provides richer information than binary answers, which leads to better prediction accuracy. On the other hand, rating scale may take more user time than binary answers.

7. REFERENCES

[1] D. Agarwal and B. Chen. Regression-based latent factor models. In *Proc. of the ACM SIGKDD*, pages 19–28. ACM, 2009.

[2] C. Boutilier, R. Zemel, and B. Marlin. Active collaborative filtering. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, pages 98–106, 2003.

[3] J. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proc. of Uncertainty in Artificial Intelligence*, 1998.

[4] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[5] N. Golbandi, Y. Koren, and R. Lempel. On bootstrapping recommender systems. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 1805–1808. ACM, 2010.

[6] N. Golbandi, Y. Koren, and R. Lempel. Adaptive bootstrapping of recommender systems using decision trees. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 595–604. ACM, 2011.

[7] A. Gunawardana and C. Meek. Tied boltzmann machines for cold start recommendations. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 19–26. ACM, 2008.

[8] A. Harpale and Y. Yang. Personalized active learning for collaborative filtering. In *Proc. of ACM SIGIR Conference*, pages 91–98. ACM, 2008.

[9] L. He, N. Liu, and Q. Yang. Active dual collaborative filtering with both item and attribute feedback. In *AAAI*, 2011.

[10] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *Proc. of ACM SIGIR Conference*, 1999.

[11] T. Hofmann. Latent semantic models for collaborative filtering. *ACM Transactions on Information Systems*, 22(1):115, 2004.

[12] 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*, pages 278–285, 2004.

[13] Y. Koren. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Transactions on Knowledge Discovery from Data*, 4(1):1–24, 2010.

[14] N. D. Lawrence and R. Urtasun. Non-linear matrix factorization with gaussian processes. In *Proc. of the ICML*, 2009.

[15] D. M. Pennock, E. Horvitz, S. Lawrence, and C. L. Giles. Collaborative filtering by personality diagnosis: A hybrid memory- and model-based approach. In *Proc. of the Conference on UAI*, 2000.

[16] P. Pu and L. Chen. User-involved preference elicitation for product search and recommender systems. *AI Magazine*, 29(4):93, 2009.

[17] A. Rashid, I. Albert, D. Cosley, S. Lam, S. McNee, J. Konstan, and J. Riedl. Getting to know you: learning new user preferences in recommender systems. In *Proceedings of the 7th international conference on Intelligent user interfaces*, pages 127–134. ACM, 2002.

[18] A. Rashid, G. Karypis, and J. Riedl. Learning preferences of new users in recommender systems: an information theoretic approach. ACM, 2008.

[19] J. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *Proc. of the International Conference on Machine Learning*, 2005.

[20] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proc. of the Conference on CSCW*, 1994.

[21] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *Proc. of the International Conference on World Wide Web*, 2001.

[22] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In *Proc. of ACM SIGIR Conference*, pages 253–260. ACM, 2002.

[23] M. Schmidt. *Graphical Model Structure Learning with L1-Regularization*. PhD thesis, The University of British Columbia, 2010.

[24] M. Sun, G. Lebanon, and P. Kidwell. Estimating probabilities in recommendation systems. In *Proc. of the International Conference on Artificial Intelligence and Statistics*, 2011.

[25] S. Yang, B. Long, A. Smola, and H. Zha. Collaborative-competitive ltering: Learning recommender using context of user choice. In *Proc. of ACM SIGIR Conference*, 2011.

[26] K. Yu, S. Zhu, J. Lafferty, and Y. Gong. Fast nonparametric matrix factorization for large-scale collaborative filtering. In *Proc. of ACM SIGIR Conference*, 2009.

[27] K. Zhou, S. Yang, and H. Zha. Functional matrix factorizations for cold-start recommendation. In *Proc. of the ACM-SIGIR conference*, pages 315–324, 2011.