Interview Process Learning for Top-N Recommendation

Fangwei Hu, Yong Yu
Dept. of Computer Science and Engineering
Shanghai Jiao Tong University
800 Dongchuan Road, Shanghai China, 200240
{hufangwei,yyu}@apex.sjtu.edu.cn

ABSTRACT

In the field of recommendation system research, a key challenge is how to effectively recommend items for new users, a problem generally known as cold-start recommendation. In order to alleviate cold-start problem, recently systems try to get the users' interests by progressively querying users' preference on predefined items. Constructing the query process via machine learning based techniques becomes an important direction to solve cold-start problem. In this paper, we propose a novel interview process learning algorithm. Different from previous approaches which focus on rate prediction, our model is able to handle wide ranges of loss functions and can be used in collaborative ranking task. Experimental results on three real world recommendation dataset demonstrate that our proposed method outperforms several baseline methods.

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 System; Cold-start problem; Ranking; Decision tree; Functional matrix factorization

1. INTRODUCTION

With the development of Internet, recommendation systems have become a significant component in the domain of data mining. However, a key challenge for building an effective recommender system is the well-known cold-start problem, which is to recommend items to new users. While existing collaborative filtering algorithms can obtain high performance under warm-start condition, they will fail to recommend items for fresh users since the recommender system rarely knows the interests of those users. Recently cold-start problem has attracted much attention from researchers, which becomes a hot topic in the field of recommendation.

RecSys'13, October 12–16, 2013, Hong Kong, China.

http://dx.doi.org/10.1145/2507157.2507205.

An intuitive approach to solving the cold-start problem is to model new users' preferences by querying their responses adaptively in an interview process [1, 3, 4]. While there has been researches on the interview process learning, existing literatures focus learning models that optimizing root mean squared error (RMSE) for rate prediction task. While collaborative ranking is equally important as rate prediction, interview process learning for collaborative ranking is not explored.

In this paper, we propose a novel learning algorithm to learn interview questions for collaborative ranking task. The proposed method is based on factorization of the sparse user-item rating matrix, but we define the user latent factor to be the output of a function in the form of a decision tree, whose input is the users' answers to the interview queries. The learning task is effectively carried out with the coordinate descent based update algorithm and a estimation method for general loss function reduction beyond square loss. The contribution of the paper is listed as follows. (1) We propose a general framework of interview decision tree which can handle ranking task. (2) We empirically demonstrate the effectiveness of the proposed method on three real-world datasets.

Outline: In Section 2, we briefly present existing research works for cold-start collaborative filtering. In Section 3, we first introduce basic matrix factorization algorithm for collaborative filtering, then we present our method which models users' preference by constructing the interview process in the form of decision tree. Then, we test our proposed method's performance on three datasets and prove that our model can outperform other baselines in Section 4. Finally we conclude our work and discuss about several future directions in Section 5.

2. ALGORITHM

In this section, we describe our method for cold-start collaborative filtering which incorporate interview process into the matrix factorization model. The key contribution is that we propose a general framework of interview decision tree which can handle ranking task.

2.1 Matrix Factorization

Before describing our method, we first introduce basic matrix factorization algorithm for collaborative filtering. Let r_{ij} denote the rating of user i for item j, where $i = 1, 2, \ldots, N$ and $j = 1, 2, \ldots M$. We generally use matrix R to present the whole rating matrix. We define observed rating set $O = \{(i, j)|r_{ij} \text{ is observed}\}$. The goal of collaborative filtering is to predict the unknown ratings in R. One method for collaborative filtering is matrix factorization. Specifically, we assign a d-dimension vector $u_i \in \mathbb{R}^d$ for each user i and $v_j \in \mathbb{R}^d$ for each item j. The rating r_{ij} of user i for item j can be estimated by inner production of u_i and v_j, i.e. $\hat{r}_{ij} = u_i^T v_j$.

We can also rewrite the expression above in matrix format, i.e. $\hat{R} = U^T V$. Our object is to estimate the parameters U and V fitting the training data by solving the following optimization problem:

$$\min_{U,V} \mathcal{L}(R, \hat{R}) + Reg(U, V) \tag{1}$$

where $\mathcal{L}$ is a loss function that measuring the difference between the prediction $\hat{R}$ and the target R. $Reg(U, V)$ is the regularization term. For the loss function $\mathcal{L}$, it can be square loss, logistic loss, pairwise rank loss, etc.

2.2 Layer-wise Functional Matrix Factorization

In the basic matrix factorization model described in Section 3.1, the user latent factor u_i is learned by optimizing the loss on the observed data. However, in order to model the new user's latent factor, we propose to search a function which can map the new user's responses into latent space. Specifically, assume we would like to ask users k interview questions, i.e. show k items to users in proper order to ask for their ratings. We also assume an answer to a question can be positive value presenting the rating or zero which means "Unknown". Furthermore, let x_i denote the k-dimension vector representing the answer of user i to the k questions. And we associate x_i with u_i by assuming $u_i = f(x_i)$, where f is a function which maps the answer vector $x_i \in \mathbb{R}^k$ to the user latent vector $u_i \in \mathbb{R}^d$. Then we can rewrite the rating estimation the user i give for item j as $\hat{r}_{ij} = v_j^T f(x_i)$.

Our goal is to learn both f and v_j from the training data. By substituting $u_i = f(x_i)$ into 1, we have the following optimization problem:

$$\min_{f \in \mathcal{F}, V} \mathcal{L}(R, f(X)^T V) + \lambda \|V\|^2 + \lambda \Omega(f) \tag{2}$$

where $f(X) = (f(x_1), \ldots, f(x_N))$ is the matrix of all item latent factors mapped from their response vectors. And $\mathcal{F}$ is the function space from which $f(x)$ is selected and the second term is the regularization term in which $\|\cdot\|$ represent Frobenius norm. $\Omega(f)$ defines the complexity of the function f.

Furthermore, our system should query adaptively, that is follow-up questions should be selected based on the user's response to the previous questions. What's more, since we allow the user to answer "Unknown", our system need to be capable with this situation. Following prior works [2, 4, 5], we use a decision tree to represent function $f(x)$, i.e. the function space $\mathcal{F}$ is decision tree space. Specifically, each node of the decision tree represents an interview question, and it has three branches which mean "higher than threshold", "lower than threshold" and "Unknown", respectively. After the user answer a series of questions, we can obtain a path from the root node to the leaf node according to the user's responses.

We further utilize the decision tree structure to encourage the latent factors of each nodes close to their parents. To achieve this, we assign a preference vector to each node in the decision tree, and define the latent factor of each node to be the sum of all the vectors along the path to the root. This can be formally expressed by following equation

$$f(x_i) = \sum_{s \in path(x_i)} \Delta u_s \tag{3}$$

where $path(x_i)$ gives the node on the path that corresponds to answers x_i in the tree. We use symbol Δu_s to emphasize that each parameter is the difference between current nodes' latent factor to its parent's latent factor. We further define the $\Omega(f)$ as

$$\Omega(f) = \sum_s |\Delta u_s|^2 \tag{4}$$

This allows the optimization step to encourage the latent factor of each node to be close to its parent. We call our model layer-wise functional matrix factorization model. Figure (1) gives the outline of our methods.

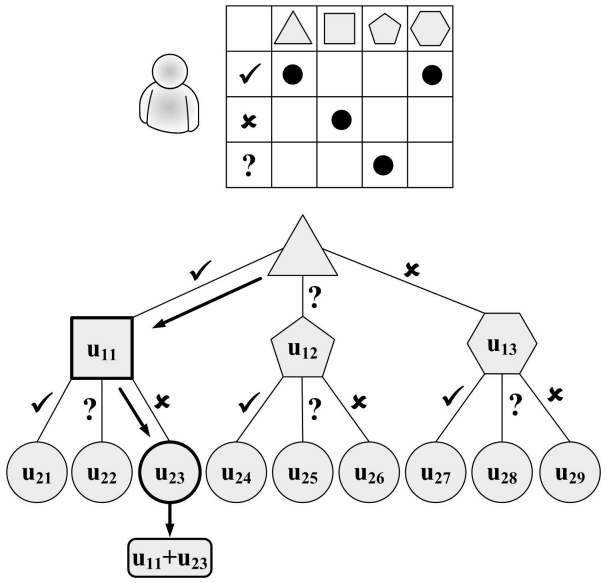


Figure 1: Layer-wise Functional MF

2.3 Alternating Optimization

The objective function defined in 2 can be optimized through an alternating minimization process. Specifically, we can partition all parameters in Equation (2) into two parts and then optimize them separately between the following two steps:

1. Fix $f(x)$, we can update v_j by regularized coordinate descent. According to the objective function 2, we update one parameter p among all parameters V for each iteration:

$$p = \operatorname*{argmin}_p \mathcal{L}(R, f(X)^T V) + \lambda \|V\|^2 \tag{5}$$

2. Fix v_j, we try to learn a function f formed in decision tree s.t.

$$f = \operatorname*{argmin}_{f \in \mathcal{F}} \mathcal{L}(R, \hat{R}) \tag{6}$$

The most significant problem now is how to find the optimal decision tree from a large function space. Since it is computationally impossible for us to find out the global optimal function, we propose a layer-wise additive update with an efficient greedy update algorithm to find a approximate solution instead.

2.4 Layer-wise Additive Training

In our work, the objective function is defined in Equation (2), and we now describe a greedy algorithm to construct a decision tree which fit the training data approximately best. Specifically, at each node, we select a best interview question(i.e. a best item to rate) and a best rating threshold by optimizing the loss function Equation (2); then the tree will split the user set into three parts. We repeat this procedure recursively until the tree reaches the redefined depth limit. In our experiments, the depth limit is usually set to a small number between 3 and 10.

Formally, starting from the root node, once the interview question q and the rating threshold t are fixed, we can divide the set of users at current node into three disjoint subsets $C_l(q,t)$, $C_r(q,t)$ and $C_m(q,t)$ as mentioned above. For any subset C, we would like to find out the optimal increment on the user factor,

$$\Delta u_{best} = \underset{\Delta u}{\operatorname{argmin}} \mathcal{L}(R|C, (u_P+\Delta u, \ldots, u_P+\Delta u)^T V)+\lambda_u \|\Delta u\|^2 \tag{7}$$

Here $R|C$ means that constrain R on the user set C, and $(u_P + \Delta u, \ldots, u_P + \Delta u)$ represents that make $|C|$ copies for the latent factor $u_P + \Delta u$. The vector u_P means the user latent factor obtained by parent node where $C_l \cup C_r \cup C_m$ belongs to. Our method adds a leaf node for each subset, and thus add a Δu to latent factors of each subset.

It is time consuming to directly calculate the Δu_{best} for general loss, we use a more effective approximated solution instead. The first part of the objective function can be approximated by Taylor expansion as:

$$\begin{aligned} & \mathcal{L} \quad (R|C, (u_P + \Delta u, \ldots, u_P + \Delta u)^T V) + \lambda_u \|\Delta u\|^2 \\ \simeq \; & \mathcal{L} \quad (R|C, (u_P, \ldots, u_P)^T V) + \Delta u^T (\nabla \mathcal{L} + 2\lambda_u \Delta u) \\ & + \quad \frac{1}{2} \Delta u^T (\mathbf{H} + 2\lambda_u) \Delta u \\ = \; & \widetilde{\mathcal{L}} \quad (\Delta u) \end{aligned}$$

Here $\nabla \mathcal{L}$ means gradient of loss function $\mathcal{L}$ and $\mathbf{H}$ means the Hessian matrix of $\mathcal{L}$ which is formally defined as:

$$\mathbf{H} = \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial \Delta u_1^2} & \frac{\partial \mathcal{L}}{\partial \Delta u_2 \Delta u_1} & \cdots \\ \frac{\partial \mathcal{L}}{\partial \Delta u_1 \Delta u_2} & \frac{\partial \mathcal{L}}{\partial \Delta u_2^2} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \tag{8}$$

We use $\widetilde{\mathcal{L}}(\Delta u)$ to find the best user latent factor Δu_{best}. Let $g = \nabla \mathcal{L} + 2\lambda_u \Delta u$ and $\widetilde{\mathbf{H}} = \mathbf{H} + 2\lambda_u$, and we can rewrite $\widetilde{\mathcal{L}}$ as:

$$\widetilde{\mathcal{L}}(\Delta u) = \widetilde{\mathcal{L}}(0) + \Delta u^T g + \frac{1}{2} \Delta u^T \widetilde{\mathbf{H}} \Delta u \tag{9}$$

Mathematically, the optimal Δu has a closed-form solution given by $\Delta u_{best} = -\widetilde{\mathbf{H}}^{-1} g$. Substituting the equation into 9, we can get the result that

$$\begin{aligned} \min_{\Delta u} \widetilde{\mathcal{L}}(\Delta u) &= \widetilde{\mathcal{L}}(0) - g^T (\widetilde{\mathbf{H}}^{-1})^T g + \frac{1}{2} g^T (\widetilde{\mathbf{H}}^{-1})^T g \quad (10) \\ &= \widetilde{\mathcal{L}}(0) - \frac{1}{2} g^T (\widetilde{\mathbf{H}}^{-1})^T g \quad (11) \end{aligned}$$

Having above approximate expression for the optimal value of loss function, we can decide which partition strategy is the best one. However, it is also time consuming to calculate the inverse matrix, so we use the diagonal approximation of of $\widetilde{\mathbf{H}}$. Since the inverse matrix of a diagonal matrix is easy to get, we would use the diagonal matrix to estimate Δu_{best}. Denote $\mathbf{D}$ as the diagonal matrix of $\widetilde{\mathbf{H}}$, and Equation (10) can be simplified as

$$\min_{\Delta u} \widetilde{\mathcal{L}}(\Delta u) \simeq \widetilde{\mathcal{L}}(0) - \frac{1}{2} g^T \mathbf{D}^{-1} g \tag{12}$$

We can use Equation (12) to find the optimal interview question q and the best threshold t. Note since there are approximations in the derivations, we only use it to find the interview question. After the question is found, we use coordinate descent to obtain optimal Δu. The update process will enumerate every dimension of Δu_i and fix other dimensions to optimize Δu_i. After the root node is constructed, its children can be constructed in a similar way recursively until the depth of the tree reaches the depth limit.

3. EXPERIMENTS

In this section, we design a set of experiments on three benchmark datasets to make comparison between our method and other baselines.

3.1 Datasets and Experiments Setting

Firstly, we describe our three datasets which are used in experiments.

- The MovieLens 1M dataset contains 6040 users, 3952 movies and about 1 million ratings.
- The Netflix is a large dataset for testing collaborative filtering algorithm. It contains 480189 users, 17770 items and over 100 million ratings.
- The Yahoo! Music KDDCup Track 1 dataset which contains 253 million ratings includes 1 million users and over 624 thousand music from the Yahoo! Music website.

We test the ranking performance in our experiments. The performance of ranking task can be measured by mean average precision (MAP), which is defined as follows

$$MAP = (\frac{1}{|\mathcal{U}|} \sum_{i \in \mathcal{U}} (\frac{1}{|\mathcal{O}_i|} \sum_{j \in \mathcal{O}_i} P@(rank_i(j))) \tag{13}$$

where $\mathcal{U}$ represents the set of all users included in test set, $\mathcal{O}_i$ means the observed item set of user i and $rank_i(j)$ calculates the rank of item j in the ranking list of user i. Furthermore, $P@n$ calculates the fraction of top n recommended items that are positive.

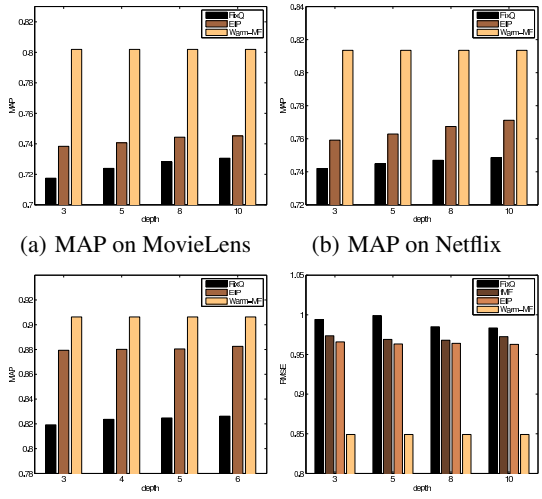
Our experiment setting is as same as that in Zhou et al. [5]. For each dataset, we split the users into two disjoint subsets, the training set and the test set. The ratings of each user in the test set are partitioned into two sets: the first set is used to generate the user responses while the second set is used to evaluate the performance. For ranking task, our experimental setting here is to sample negative instances for training data and test data separately. For test data, to simulate users' rank list, we randomly sample negative items 10 times as positive items for each user as test data.

The algorithms tested in our experiments are summarized as follows,

- **EIP** It is our algorithm which is short for efficient interview process. It can handle various kinds of loss function including rank loss, and it has excellent efficiency. We set λ_u at 10 in our experiments by cross validation.
- **FixQ** It is short for "fixed questions" which represents the method that select top-k popular items as static interview questions, and then use feature based MF to solve the problem by leverage the answers to interview as implicit feedback.
- **Warm-MF** Additionally, we combine training set and answer set together as a new training set to simulate the situation of warm-start matrix factorization.
- **fMF** It is proposed in Zhou et al. [5], in which they use closed-form solution to optimize loss function alternately. Although it cannot handle other loss function but square loss, for completeness, we also conduct experiment on rate prediction and use it as a baseline.

3.2 Results on Ranking

In this section, we test the ranking performance of our algorithm on three datasets.



(a) MAP on MovieLens (b) MAP on Netflix

(c) MAP on Yahoo! Music (d) RMSE on MovieLens

Figure 2: Experimental Results

3.2.1 MAP Performance on MovieLens

Firstly, we evaluate the MAP for EIP, FixQ and Warm-MF on MovieLens since fMF cannot handle the logistic loss. We test four sets of results by setting number of interview questions at 3, 5, 8, 10 respectively. The detailed MAP results of three algorithms are shown in Figure (2(a)). From the MAP results of each of the algorithm, we can get some basic observations. Firstly, we find that our proposed approach EIP do outperforms FixQ in this rank task. And MAP of EIP is higher than FixQ by 2% averagely. Secondly, Warm-MF works better than EIP. And the gap between these two algorithm is quite large, which is about 6%. That is, on this dataset, the advantage of warm-start situation is obvious. Finally, the improvement of MAP caused by the increment of the number of interview question is quite notable.

3.2.2 MAP Performance on Netflix

In this section, we report the performance of three algorithms on Netflix. We give the comparison of our method against other two in Figure (2(b)). And here we also set the number of interview questions at 3, 5, 8, 10, which is similar to that in previous sections. It is evident that EIP achieves a significant improvement over FixQ. Here the MAP of EIP is still higher than that of EIP by about 2%. The warm-start situation also has obvious advantage, which outperform EIP by 5% approximately. Furthermore, the effect of the increment of the number of interview questions is still evident here.

3.2.3 MAP Performance on Yahoo! Music

Finally, experiments are conducted on Yahoo! Music dataset, the largest dataset mentioned in this paper. Different from previous sections, we set the number of interview questions at 3, 4, 5, 6 to make the training time shorter because of the scale of the dataset. We present MAP results of three algorithm as Figure (2(c)). Figure (2(c)) gives us an abstract impression that EIP outperform FixQ obviously. The gap between EIP and FixQ reaches about 7% averagely. And another bright spot in Figure (2(c)) is that EIP outperforms Warm-MF by a bit less than 3%, which means our method almost reaches the rank performance of warm-start situation in this dataset. Last but not the least, the improvement caused by the depth of the decision tree is not as evident as above. Even when the depth reaches 6, the MAP begins to decrease, which means our model becomes overfitting.

3.3 Results on Prediction

In this subsection, we show more experimental results of our algorithm on prediction tasks and we use widely used metric RMSE as evaluate the performance of each algorithm.

We compare the prediction performance on MovieLens. We also involve fMF here to make a comparison with other two algorithms. As Figure (2(d)) shows, our method outperforms fMF and FixQ. However, we can also find that the RMSE performance of cold-start algorithms is far worse than that of Warm-MF, which is not similar to MAP performance. Furthermore, when the depth becomes deeper, the results of fMF and EIP show signs of deterioration. We think that this phenomenon is caused by overfitting since the skewness of small dataset.

4. CONCLUSIONS

The main focus of this paper is on the cold-start problem in recommender systems. We have proposed a framework for effectively learning a decision tree to handle different kinds of tasks including ranking task. We can simulate interview process to model the latent factors for new-come users. Furthermore, our framework is able to handle different types of loss function; even they do not have closed form solution of the optimal point. We have established learning algorithm based on alternating optimization. At last, we have demonstrated the effectiveness of our method on real-world recommendation benchmarks and compared our method with other baselines.

The current model make the use of Taylor expansion and some inequality estimation to speed our method up and it may cause the harm to the performance. For future work, we plan to investigate how to make the update rules used in coordinate descent more precise, as well as the expected optimal value of loss function. Moreover, we also plan to apply more kinds of loss function for different tasks.

5. REFERENCES

[1] N. Golbandi, Y. Koren, and R. Lempel. On bootstrapping recommender systems. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, CIKM '10, pages 1805–1808, New York, NY, USA, 2010. ACM.

[2] 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*, WSDM '11, pages 595–604, New York, NY, USA, 2011. ACM.

[3] 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 *Proceedings of the 7th international conference on Intelligent user interfaces*, IUI '02, pages 127–134, New York, NY, USA, 2002. ACM.

[4] A. M. Rashid, G. Karypis, and J. Riedl. Learning preferences of new users in recommender systems: an information theoretic approach. *SIGKDD Explor. Newsl.*, 10(2):90–100, Dec. 2008.

[5] 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*, pages 315–324. ACM, 2011.