Interactive Recommendation

Yang Yang and Jianfei Wang
Deparment of Computer Science and Technology, Tsinghua University
{y-yang-11, jf-wang12}@mails.tsinghua.edu.cn

1. INTRODUCTION

In some web applications, users' needs are vague. They do not know what exactly they want to retrieve. Some methods like recommendation methodologies aim to solve this issue. In this work, we aim to help users identify their needs by asking them as few as possible "Yes or No" questions, e.g., "Are you looking for a shopping mall?", "Do you prefer to have your internship in a software company or not?", etc. Ideally, each question divides the hypothesis space in half and find the target object. We call this methodology as interactive recommendation as it requires the user's response and need to interact with the user during the progress.

On the other hand, for some mobile-end users, entering a query costs their operations and is not so convenient. As an example, when one is in a bus and is looking for a football game result of last night, instead of typing "Barcelona" to the search engine, he or she may prefer to click just few radio buttons ("Yes" or "No") and get the answer.

2. PROBLEM DEFINITION

We introduce some necessary definitions and then formulate the problem. Consider we have n objects and m tags, and the user's *target object* is t. We first define a tag matrix as

Definition 1. **Tag matrix.** A tag matrix $X \in \{0,1\}^{n\times m}$ gives the relation of the objects and tags, where $x_{ij} = 1$ stands for object i has tag j.

Next we formulate the questions which interact with the user and are used to partition the hypothesis space:

Definition 2. **Yes/No question.** A Yes/No question can be regarded as a two-tuple (k, r), where k is a tag and $r \in \{0, 1\}$ stands for the user's answer. $r = 1$ if the target object t has tag k, otherwise $r = 0$.

Given a a tag matrix X and a *target object* t, our goal is to identify t in as few Yes/No questions as possible.

3. APPROACH

3.1 Framework

We use a vector $\mathbf{W}$ as the weights of all objects. All weights are initialized to 1. After the t-th round of Yes-No question, a weight $w_i^{(t)}$ will be multiplied by a *discounter* γ ($\gamma < 1$) if the corresponding object i does not "match" the question, i.e., if object i has the tag j but the user answers "no" or vice versa. The details of the framework can be found in Algorithm 1.

Next we will introduce two algorithms for finding the tag used for each question.

3.2 Greedy Algorithm

We first introduce a greedy algorithm. The basic idea is to choose the tag used in the question to maximally decreases $\|W^t\|_1$ regardless of the answer. To achieve the goal, we first define the expected value of each tag j as

$$E(j) = \|w^t\|_1^{-1} \sum_{j\in Item} X_{ij} w_j^t \tag{1}$$

Based on this, we can also calculate each tag's variance:

$$var(j) = \|w^t\|_1^{-1} \sum_{j\in Item} w_j^t (x_{ij} - E(j))^2 \tag{2}$$

As the tags are binary, we can loosely interpret each tag as a Bernoulli random variable. It can be proved $\|W^t\|_1$ will drop largely when we select tag k with the maximum variance.

3.3 Heuristic Search

Next we propose an algorithm based on heuristic search. By constructing the heuristic function carefully and applying the heuristic search method to the procedure of searching for the tag questions, we can improve the performence of the interactive recommendation.

We first show how we design the heuristic function for the case where we search for k tag questions in one iteraton.

Suppose in each search step we consider k questions used to ask the user one by one. Our goal is to reduce the $\|W\|_1$ largely in the worst case, which is similar to the greedy algorithm.

We let these k question tags be $Q_k = \{q_1, q_2, \ldots, q_k\}$, and let the answers to these question tags be $\mathbf{l} = l_1 l_2 \ldots l_k \in \{0,1\}^k$ where 1 stands for "yes" and 0 stands for "no".

Assume that after answering the m-th question tag, the weight for the i-th item is $w_i^{(m)}$, and the answer for the $(m+1)$-th question is "yes", then after answering the $(m+1)$-th question the weight for the i-th item becomes

$$w_i^{(m)} \gamma^{1-x_{i(m+1)}}; \tag{3}$$

If the answer is "no", after answering the $(m+1)$-th question

Input:
a tag matrix X, the maximum number of questions T, and the discounter γ.
Output:
a updated weight vector $W^{(T)}$.

Algorithm:
initialize weights $w_i^{(0)} = 1$ for all i;
t = 0 ;
while $t < T$ **do**
  q = QuestionSelectionAlgo(X, W^t, γ);
  Ask the user if q is present in the target object;
  if *the answer is "yes"* **then**
    $w_i^{(t+1)} = w_i^{(t)} \gamma^{1-x_{iq}}$ for all i
  else
    $w_i^{(t+1)} = w_i^{(t)} \gamma^{x_{iq}}$ for all i
  end
  $t = t + 1$
end
Return $W^{(T)}$

Algorithm 1: The interactive recommendation framework.

the weight becomes

$$w_i^{(m)} \gamma^{x_{i(m+1)}} \tag{4}$$

In summary, assume that the answer is $l \in \{0, 1\}$, after answering the $(m + 1)$-th question the weight becomes

$$w_i^{(m)} \gamma^{l \oplus x_{i(m+1)}} \tag{5}$$

Therefore after answering all these k question tags, the weight for the i-th item is:

$$w_i \prod_{j=1}^{k} \gamma^{l_j \oplus x_{ij}} \tag{6}$$

Hence the total cut weight is

$$W[Q|\mathbf{l}] = \sum_i w_i (1 - \prod_{j=1}^{k} \gamma^{l_j \oplus x_{ij}}) \tag{7}$$

And since $\mathbf{l} \in \{0, 1\}^k$, and $\mathbf{l}$ can be any 0-1 string with k entries, it follows that there are in total 2^k different answer strings, i.e., 2^k different $\mathbf{l}$'s may appear as the answer string of the user. Thus our question should be selected as:

$$Q^* = \arg\max_Q \min_{\mathbf{l} \in \{0,1\}^k} \{\sum_i w_i (1 - \prod_{j=1}^{k} \gamma^{l_j \oplus x_{ij}})\} \tag{8}$$

We call a heuristic search algorithm to find Q^*. With the optimization of calculate the heuristic function for each possible Q, the time complexity of this algorithm can be proved as $O(m^k \times C)$, where $C = \max_j \{\sum_i X_{ij}\}$.

The point is that, even though we search for k tag questions in one iteration, we only pick one of them as the tag question and present it to the user. Thus in each iteration the user still answers one question.

If we let $q_1, q_2, \ldots, q_t$ be the sequence of questions that user answers, and after answering q_t the algorithm stops, we will have α as the lower bound of the ratio of the cut weight for all these tag questions during the procedure of searching for the target object, namely

$$\alpha = \min_{1 \le j \le t} \min_{l \in \{0,1\}} \frac{W^{(j)}}{w_i^{(j)}(1 - \gamma^{l \oplus x_i})}$$

Then we have the following theorem:

Theorem 1 : Let p be the probability that the user answers incorrectly, and γ be the discounter. Also we use a string of random variables y_i to denote whether the user answers incorrectlyl in the i-th iteration. If the user answers incorrectly in the i-th iteration, $y_i = 1$, otherwise $y_i = 0$. Namely

$$\Pr[y_i = 1] = p$$

Let $Y_t = \sum_{i=1}^{t} y_i$.
Then if t and Y_t meet the condition that

$$t(\ln(\gamma)\frac{Y_t}{t} - \ln[1 - \alpha(1 - \gamma)]) = y > 0$$

Then the target object will be surely in top L objects with highest weights where $n \ge L \ge \frac{n}{e^y}$.

Moreover, if we have

$$p \le \gamma$$

Then in expection, the target object will be surely in top L objects with highest weights where $n \ge L \ge n(\frac{1-\alpha(1-\gamma)}{1-p(1-\gamma)})^t$

Proof :

Since in t iterations the user answers Y_t questions incorrectly, the weight of the target object is γ^{Y_t}. We know that at the t-th iteration, the total weight $W^{(t)}$ is no greater than $W^{(0)}(1 - \alpha(1 - \gamma))^t = n[1 - \alpha(1 - \gamma)]^t$, because at least $\alpha(1 - \gamma)W^{(i-1)}$ vainishes at the i-th iteration.

Since $y > 0$, we have $e^y > 1$. Let $n \ge L \ge \frac{n}{e^y}$, then $y \ge \ln(n) - \ln(L)$. The the condition becomes

$$t(\ln(\gamma)\frac{Y_t}{t} - \ln[1 - \alpha(1 - \gamma)]) \ge \ln(n) - \ln(L)$$
$$\Rightarrow L \times \gamma^{Y_t} \ge n[1 - \alpha(1 - \gamma)]^t$$

Therefore we have

$$L \times \gamma^{Y_t} \ge n[1 - \alpha(1 - \gamma)]^t \ge W^{(t)}$$

which indicates that the weight of the target object is at least the L-th highest. Therefore the target object is in the top L objects with the highest weight.

When we have

$$p \le \gamma$$

We have

$$\frac{1 - \alpha(1 - \gamma)}{1 - p(1 - \gamma)} \le 1$$

Thus when $n \ge L \ge n(\frac{1-\alpha(1-\gamma)}{1-p(1-\gamma)})^t$, in expectation we have

$$\begin{aligned}
\mathbb{E}[L \times \gamma^{Y_t}] &= L \times \mathbb{E}[\gamma^{Y_t}] \\
&= L \times \prod_{i=1}^{t} \mathbb{E}[\gamma^{y_i}] \\
&= L \times [1 - (1 - \gamma)p]^t \\
&\ge n[1 - \alpha(1 - \gamma)]^t \\
&\ge W^{(t)}
\end{aligned}$$

Table 1: The performance of different algorithms in ArnetMiner data set.

Probability of lying	IHS	SHS	Greedy
0.00	34.24	34.08	35.08
0.02	27.54	26.98	27.16
0.04	28.88	27.86	31.1
0.06	37.87	36.8	38.8
0.08	35.18	35.2	34.32
0.10	43.8	40.9	43.14

4. EXPERIMENTS

We perform experiments on 3 data sets to compare the performance of our approach (IHS), greedy, and static heuristic search (SHS). To show the power of interaction with the user, we propose the third algorithm SHS. The question selection method of this algorithm is the same as IHS, but it only interacts with the user every 2 rounds, in one iteration we search for $k = 2$ question tags, use both of them as question tags and present them sequentially to the answerer. Note that during these 2 iterations, the system doesn't have any interaction with the user, i.e., the answer of the user in the first iteration doesn't infect how the system choose the question tag in the second iteration.

We conduct an experiment to compare the performance of our approach to baselines. The basic setup of the experiment is as follows: we first select an object as the target. In each turn, given the tag matrix, the question selection algorithm will present a question tag q to the answerer. We then build simulators of answerers. When the answerer is asked by question q, it will reply "Yes" if tag q is present in the target object, or "No" otherwise. The interactive progress continues until the target object ranks in the first place, and no other object ranks abreast of the target.

The data set consists of 36,371 conferences in computer science and 1,905,496 papers provided by ArnetMiner[1]. We then generate distributions θ_v for each conference v on 200 topics by ACT [7]. We treat each conference as an item and each topic as a tag. We set the entity in tag matrix $x_{vz} = 1$ if $\theta_{vz} > \beta$, where z is a topic and β is a threshold manually defined. In the tag matrix, each row contains 21.2 entities with value one on average.

Table 1 shows the the average number of questions need to be generated to find the target item by different algorithms.

5. PROTOTYPE SYSTEM

We have developed and deployed a web application for interactive user intention understanding based on our approach and Patent data set in PatentMiner[2]. Figure 1 shows a screenshot of the prototype system. Users can find suitable companies for job hunting or business analysis. From experimental results, we can see that in Patent data set, IHS requires around 14 questions to identify the target. But in real applications, it uses less questions as the system displays top 5 companies during each round, thus the user can find the target even when it does not rank the first. Throughout the experiments, we find that the questioner averagely requires 2-3 questions to make an item ranks first from top 5. Also, the system allows users to enter a query and search for some candidates first to limit the size of hypothesis space and reduce the number of questions required. At present, we've started collecting users' feedbacks from the prototype system to further improve our work.

[1]http://arnetminer.org/

[2]http://pminer.org

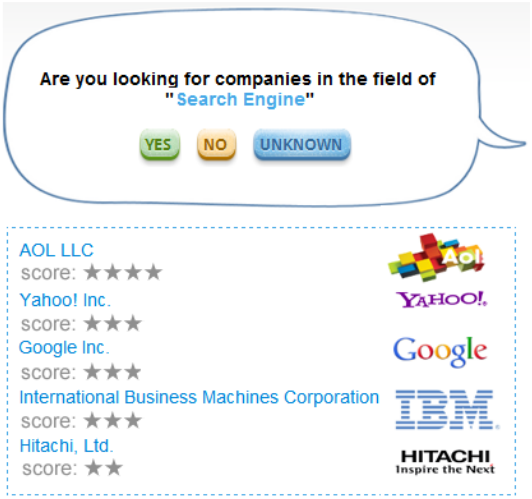

Figure 1: A screenshot of the prototype system.

6. RELATED WORK

To understand users' intentions, various issues based on search engines and recommendation systems were developed.

Link analysis [3, 5] is a data-analysis technique used to evaluate relationships between nodes in the network. Techniques like this will help the system find out the popular items, which will in turn help users search for valuable items and information.

Many recommendation engines were also developed. Typically, users need to register and tell the recommendation engines their preferences on some items explicitly, since most of traditional recommendation methods rely heavily on user logs and feedbacks [4].

In [6], authors make analysis on different item-based recommendation generation algorithm, including the computation of similarities between items and the way of obtaining recommendations. Meanwhile, they made comparisons between their results and basic k-nearest neighbor approach [2, 1], which is based on collaborative filtering and is a popular recommendation system.

However, recommendation engines may not produce meaningful recommendations when users can not express their preferences accurately or there is no available user logs.

7. REFERENCES

[1] T. Denoeux. A k-nearest neighbor classification rule based on dempster-shafer theory. *Classic works of the Dempster-Shafer theory of belief functions*, pages 737–760, 2008.

[2] S. Dudani. The distance-weighted k-nearest-neighbor rule. *Systems, Man and Cybernetics, IEEE Transactions on*, (4):325–327, 1976.

[3] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, Sept. 1999.

[4] D. Lemire and A. Maclachlan. Slope one predictors for online rating-based collaborative filtering. In *Proceedings of SIAM Data Mining (SDM'05)*, 2005.

[5] R. Lempel and S. Moran. Salsa: the stochastic approach for link-structure analysis. *ACM Trans. Inf. Syst.*, 19(2):131–160, Apr. 2001.

[6] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 285–295. ACM, 2001.

[7] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su. Arnetminer: extraction and mining of academic social networks. In *KDD'08*, pages 990–998, 2008.