Distributed Recommender Profiling and Selection With Gittins Indices

Li-Tung Weng Yue Xu Yuefeng Li Richi Nayak
School of Software Engineering and Data Communications
Queensland University of Technology, QLD 4001, Australia
soloman1124@hotmail.com, {yue.xu, y2.li, r.nayak}@qut.edu.au

Abstract

Most existing recommender systems nowadays operate in a single organizational base, and very often they do not have sufficient resources to be used in order to generate quality recommendations. Therefore, it would be beneficial if recommender systems of different organizations can cooperate together to share their resources and recommendations. In this paper, we present a distributed recommender system model that consists of multiple recommender systems from different organizations. With the hope to provide better recommendation service to users, the recommender systems can improve their performances by sharing their recommendations cooperatively. A recommender selection technique based on the Gittins indices [4] is presented in this paper, and it makes selections based on the stability, average performance and selection frequency of the recommenders.

1. Introduction

In this paper we proposed a recommender profiling scheme and a recommender selection algorithm that utilizes the proposed profiling scheme to select suitable recommender peers. The recommender selection problem is modeled as the classical exploitation vs. exploration (or k-armed bandit) problem [4, 1], in which the recommender selection has to be balanced between choosing the best known recommender peers to keep users satisfied and selecting other unfamiliar recommender peers to obtain knowledge about them. The proposed recommender selection algorithm is based on evaluating the Gittins Indices[4] for every recommender peers, and the indices reflect the average performance, stability and selection frequency of the recommenders.

2. Distributed Recommender Profile

2.1. System Model

We envision a world populate with a finite set of users \( U \) and a finite set of items \( I \). The proposed distributed recommender system is denoted as \( \Phi \) containing a set of \( n \) recommender peers \( R_1, R_2, ..., R_n \) (i.e. \( \Phi = \{ R_1, ..., R_n \} \)). The number of recommender peers is much smaller than the number of users in our system; specifically, \( n \ll |U| \). The set of users that uses \( R_i \) is given by \( U_i \subseteq U \), and \( R_i \)'s item set is given by \( I_i \subseteq I \), where \( U = \bigcup_{R_i \in \Phi} U_i \) and \( I = \bigcup_{R_i \in \Phi} I_i \). Moreover, in our system some users and items can be owned by more than one recommender peers such that:

\[
\bigcup_{R_i \in \Phi} U_i \cap U_j \neq \emptyset \quad \text{and} \quad \bigcup_{R_i \in \Phi, R_i \neq R_j} I_i \cap I_j \neq \emptyset
\]

2.2. User Clustering

Intuitively, a large set of users can be separated into a number of clusters based on the user preferences. Since users within the same cluster usually share similar tastes [2] and a cluster with a large number of users and a high degree of intra-similarity can better reflect the potential preferences of the users belonging to the cluster, a collaborative filtering based recommender can improve its recommendation quality by searching similar users within clusters rather than the whole user set [6]. However, different user clusters often vary in quality. The performance of such clustering based collaborative filtering system is strongly influenced by the quality of the clusters [6]. For a given recommender, some users might be able to receive better recommendations if they belong to a cluster with better quality (the cluster has a large number of users and a high intra-similarity), whereas some other users may not be able to get any constructive recommendations because the cluster to which they belong is small and has a low intra-similarity.
This situation is closely related to the well-known cold-start problem which happens when a recommender makes recommendations based on insufficient data resources. Therefore, even for the same recommender, the recommendation performance might be different for different groups of users if different user clusters have different quality. In order to provide good recommendations to various users, the proposed distributed recommender system allows its recommenders to choose recommender peers for recommendations to the current user based on the peers’ performance to a particular user cluster to which the current user belongs. We expect this design to solve the cold start problem because a recommender which is making recommendations to a user who belongs to a weak cluster can get recommendations from peer recommenders which have performed well to that group of users.

In the proposed system, every recommender peer has its own set of user clusters, we denote the set of user clusters owned by $R_i$ as $C_i$, $C_i = \{C_{i1}, \ldots, C_{im_i}\}$, $C_{ij} \subset U_i$ and $C_{ij} \cap C_{ik} = \emptyset$ for $j \neq k$, $1 \leq j \leq m_i$, $1 \leq k \leq m_i$. Because different recommenders have different user sets and different clustering techniques, the size of their cluster set might vary as well, i.e. $\exists R_i, R_j \in \Phi : |C_i| \neq |C_j|$.

### 2.3. Recommender Profile

In this section, we present our approach to profile the recommender peers within the distributed recommender system. To begin with, the performance evaluation of the recommender peers is explained. The performance of a recommender is measured by the degree of user satisfaction to the recommendations made by the recommender [5, 7, 3]. In a narrower perspective, the performance of a recommender equals to the differences between the recommender predicted user preferences and the actual user feedback. In our system, a recommender Peer $R_i$ makes recommendations to a user with a set of $k$ items $\hat{I}_i = \{i_{i1}, i_{i2}, \ldots, i_{ik}\}$ where $\hat{I}_i \subset I_i$. Once receiving the recommendation, the user then input his or her evaluations to each of the $k$ items. We use $\gamma_j$ to denote the user’s rating to item $i_j$. The value of $\gamma_j$ is between 1 and 0 which indicates how much the user likes item $i_j$, $\gamma_j = 1$ indicates the user highly prefers the item. Hence, each time a recommender peer generates a recommendation list (e.g. $\hat{I}_i$) to a user, it will get feedback $\Gamma = \{\gamma_{i1}, \gamma_{i2}, \ldots, \gamma_{ik}\}$ from the user, where $0 < \gamma_j < 1$. With $\Gamma$, we can compute the recommender peer’s current performance $\chi$ to the user by:

$$\chi = \frac{\sum_{\gamma \in \Gamma} \gamma}{|\Gamma|}$$  (1)

Formula (1) measures the performance of the recommender to this particular user in this time around. We can use the average performance of the recommender to the users in the same cluster to measure its performance to this group of users. The average performance measures how well the recommender averagely performed in the past. However, the average performance doesn’t reflect whether the recommender is generally reliable or not. Hence, we employed the standard deviation technique to measure the stability of the recommender. Another factor that should be taken into account for profiling a recommender is the selection frequency which indicates how often the recommender has been selected before. In our system, we profile each recommender peer from three aspects: recommendation performance, stability, and selection frequency. As mentioned previously in this paper, a recommender will seek for recommendations from other peers when it receives a request from a user. Broadcasting the user request to all peers is one solution, but obviously it is not a good solution since not all of the peers are able to provide high quality recommendations. In our system, the recommender will select the most suitable peers for recommendations based on their profiles. Therefore, each recommender in the distributed recommendation system keeps a profile to each of the other recommender peers.

A recommender peer may perform differently to different user clusters. Therefore its performance to different user clusters are different. For recommender $R_i \in \Phi$ which has $n$ user clusters, i.e., $C_i = \{C_{i1}, \ldots, C_{in}\}$, we use $P^{i}_{jh}$ to represent the average performance of peer $R_j$ to $R_i$’s user cluster $C_{ih}$. Hence, we can use an $m \times n$ matrix $P^i = \{P^{i}_{jh}\}_{mn}$ to represent the average performance of each of the other peers to each of $R_i$’s user clusters, where $m = |\Phi| - 1$ and $n = |C_i|$. $P^i$ is called the peer average performance matrix of $R_i$. Similarly, we use $S^i$ and $F^i$ to represent the stability and selection frequency of other peers to $R_i$. $S^i = \{S^{i}_{jh}\}_{mn}$ and $F^i = \{F^{i}_{jh}\}_{mn}$ are called the peer stability matrix and peer selection frequency matrix, respectively.

Initially, the $P^i$, $S^i$ and $F^i$ of $R_i$ are all zero matrices, because $R_i$ has no knowledge about the other peers. These matrices will be updated when a recommender peer $R_j$ helped $R_i$ to make a recommendation $\hat{I}_i$ for a user belonging to (or being classified to) a $R_j$’s user cluster $C_{ij}$. Ideally, $\hat{I}_i$ is supposed to be a subset of $I_i \cup I_j$ because the recommendation is a cooperated effort, however we assume $R_i$ entirely delegates the recommendation task to $R_j$ (i.e. $\hat{I}_i \subset I_j$) for the simplicity in this paper. After $R_i$ helped making the recommendation to the user, $R_i$ will get a feedback list (i.e. the actual user ratings) $\Gamma$ about $\hat{I}_i$ from the user. With the user feedback $\Gamma$, Formula (1) will be used to compute $\chi$ which is the $R_i$’s observation about $R_j$’s performance on $C_{ih}$. The methods for updating the average quality, stability and selection frequency matrices are given.
below respectively:

\[
P_{jh}^i = \frac{P_{jh}^i \times F_{jh}^i + \chi}{F_{jh}^i + 1},
\]

\[
S_{jh}^i = \begin{cases} 
0 & \text{if } F_{jh}^i < 1 \\
\sqrt{\frac{(P_{jh}^i - 1) \times S_{jh}^i}{F_{jh}^i}} + \frac{(\chi - P_{jh}^i)^2}{F_{jh}^i + 1} & \text{otherwise},
\end{cases}
\]

\[
F_{jh}^i = F_{jh}^i + 1
\]

where \( P_{jh}^i \), \( S_{jh}^i \), and \( F_{jh}^i \) indicate the updated matrices. The formulas described in (2) simply keep track of the average and standard-deviation of the recommender performances as well as the number of times the recommenders were selected.

### 3. Distributed Recommender Selection

#### 3.1. Gittins Indices

In this section, a brief explanation of the Gittins indices is given. The Gittins indices \([4]\) is developed to solve the k-armed bandit problem which deals with a slot machine with \( k \) arms. An amount of reward will be given when an arm is pulled. However, in each time period, only a limited number of arms can be pulled (normally one arm). Different arms have different reward distributions, and the reward distributions for the arms are initially unknown. The objective is to choose which arms to pull that will maximize the total rewards over time based on previous experience and obtained rewards as well. Formally, the k-armed bandit problem is to schedule a sequence of pullings maximizing the expected present values of

\[
E \sum_{i=1}^{\infty} a^i R(t)
\]

where \( t \) indicates the time points, \( R(t) \) denotes the sum of the rewards obtained by pulling a set of arms at \( t \), and \( a \) is a fixed discount factor where \( 0 < a < 1 \) \([8]\).

Traditionally, dynamic programming was the preferred framework for solving the bandit problem. It requires analysis of all possible combinations of the pulling sequences. However, Gittins has developed a solution in 1972 that requires computation only on the states of the individual arms \([8, 4]\). Gittins suggests to compare each potential actions (pulls) against a reference arm with a known and constant reward \( \lambda \) instead of to compare all possible actions against each other \([1]\). Gittins proved it is optimal to select actions with expected rewards equal to the reference actions with the highest equivalent rewards (i.e. index values) for each pull \([1]\). In this paper, we employed one of Gittins methods to find the index values based on the multi-population sampling in relation to the mean and standard deviation rewards of the arms. For a given discount factor \( a \), the Gittins indices can be calculated by back-solving the recurrence relation:

\[
R(\lambda, \bar{x}, \hat{s}, n) = \max \left\{ \frac{\lambda}{1-n}, \bar{x} + a \int (R(\lambda, \kappa(x), \sigma(x), n + 1)) f(x|\bar{x}, \hat{s}, n) dx \right\}
\]

where \( n \) is the current number of trials, \( \bar{x} \) is the average rewards generated from past \( n \) trials, and \( \hat{s} \) is the standard deviation of the rewards. \( \kappa(x) \) is the updated average rewards giving \( x \) is the new reward generated by the distributions function \( f(x|\bar{x}, \hat{s}, n) \) in the \( n + 1 \) trial. Respectively, \( \sigma(x) \) is the updated standard deviation of the \( n + 1 \) rewards.

Generally, (4) expresses the selection between a referenced arm with a constant reward \( \lambda \) and an uncertain arm with an expected reward \( \bar{x} \). Within the formula, the

\[
a \int (R(\lambda, \kappa(x), \sigma(x), n + 1)) f(x|\bar{x}, \hat{s}, n) dx
\]

indicates the reward obtained from the next selection (i.e. \( n + 1 \)) will be discounted by \( a \). Similarly, for the left side of the maximum function in (4), \( \frac{\lambda}{1-n} \) is the cumulative reward for always choosing the referenced arm (with the constant reward \( \lambda \)). Therefore, the Gittins index of a given arm is a value of \( \lambda \) that makes both left and right arguments of the maximum function in (4) to be equal \([1, 4]\).

Giving an arm which has been pulled for \( n \) times, and generated an average rewards \( \bar{x} \) and an standard deviation \( \hat{s} \), Gittins denotes the index value for the arm as \( v(\bar{x}, \hat{s}, n) \), and he also proved in \([4]\) that:

\[
v(\bar{x}, \hat{s}, n) = \bar{x} + \hat{s} v(0, 1, n)
\]

where \( v(0, 1, n) \) is the index value for an arm being pulled for \( n \) times with a zero average reward and a standard deviation of 1. Gittins has calculated the values of \( v(0, 1, n) \) for different combination of \( a \) and \( n \) in \([4]\). We illustrate the relation between \( n \) and \( v(0, 1, n) \) by figure 1.

Based on (5), it can be observed that as an arm’s average rewards increases, its index value increases too. Moreover, despite the average rewards, the standard deviation of the arm’s past performance and the number of times the arm has been pulled play important roles in the index calculation as well. It can be seen from Figure 1 that the standard index value \( v(0, 1, n) \) is only significant when \( n \) is small (i.e. \( n < 10 \)), when \( n \) starts getting bigger, \( v(0, 1, n) \) shrinks. By combining \( v(0, 1, n) \) with \( \hat{s} \) in (5), we can observe that the contribution of the standard deviation of an arm’s past rewards to the index value \( v(\bar{x}, \hat{s}, n) \) decreases drastically.
3.2. Recommender Selection

When a recommender $R_i$ wants to find a best recommender peer $R_j$ to make a recommendation to a user $u$, where $R_i, R_j \in \Phi$ and $u \in C_{ih}$, the following formula is used to select the most suitable peer:

$$R_j = \arg \max_{R_j \in \Phi \setminus \{R_i\}} (P^i_{jh} + S^i_{jh} \times v(P^i_{jh}))$$

where $v(n)$ is the Gittins index function that maps $n$ (i.e. selection frequency) to the corresponding $v(0, 1, n)$ based on 4. In (6), $R_i$ firstly finds the average performance, stability and selection frequency of its peers to the user cluster that $u$ belongs (i.e. $C_{ih}$). Then $R_i$ computes the index values for every peers using formula (5). In the end, the preferred peer, $R_j$ will be the one which has the highest index value.

4. Experiment Results

The detail of the experiment is omitted due to the space limitation. We employed Book Crossing dataset (http://www.informatik.uni-freiburg.de/~cziegler/BX/) for our experimentation, and our result shows that the proposed Gittins selection technique outperforms other average performances based selection techniques.

5. Conclusion

In this paper, we present a distributed recommender selection technique based on Gittins indices which is based on the past performances, stability and selection frequency of the recommenders. Based on the theory framework in [4] and our experiments, it is suggested that the recommender selection based on the Gittins method outperforms choosing recommenders simply based on the average performances.

References


