# Improve the “Long Tail" Recommendation through Popularity-Sensitive Clustering

Huaiyi Huang, Ying Zhao, Jiabao Yan, Lei Yang
Department of Computer Science and Technology / Department of Software Engineering
Tsinghua University, Beijing, 100084, China
{huanghy11, yanjb11, yanglei11}@mails.tsinghua.edu.cn
yingz@tsinghua.edu.cn

## ABSTRACT

Collaborative filtering(CF) is one of the most successful approaches to build recommender system. In recommender system, “long tail” items are considered to be particularly valuable. Many clustering algorithms based on CF designed only to tackle the “long tail” items, while others trade off the overall recommendation accuracy and “long tail” performance. Our approach is based on utilizing the item popularity information. We demonstrate that “long tail” recommendation can be inferred precisely by balanced item popularity of each cluster. We propose a novel popularity-sensitive clustering method. In terms of “long tail” and overall accuracy, our method outperforms previous ones via experiments on MovieLens, citeUlike, and MobileApp.

## Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information filtering; H.3.5 [**Online Information Services**]: Web-based services

## General Terms

Algorithms, Performance

## Keywords

Collaborative Filtering, Recommender Systems, Clustering Algorithm, Long Tail

## 1. INTRODUCTION

Collaborative filtering (CF) techniques and recommendation systems have been studied extensively in the past few decades [1, 8, 16] to fulfill users' information needs with personalized recommendations. Items with very low usage are called “long tail” items, and are very common in domains like mobile apps, reference papers, and movies. Recommendations from the “long tail” of the popularity distribution of items are generally considered to be particularly valuable [15], and are much more challenging than recommending popular items for the lack of information. Several recommendation methods were proposed in the literature with the aim of improving the “long tail” recommendation performance [10, 21, 13, 14, 22]. They are either specific recommendation methods targeting only “long tail” items [10, 21, 22], or comprehensive ones trading-off between overall recommendation accuracy and “long tail” performance [13, 14].

We propose to tackle this problem from a clustering perspective. In addition to reaching the common goals of clustering, such as grouping users with similar behaviors or to increase the density of the user-item matrix for each cluster, our clustering method also aims to make the popularity distribution of items within each cluster more balanced and to reduce the effect of “long tail”. To the best of our knowledge, this is a novel perspective that utilizes clustering collaborative filtering models to improve the “long tail” item recommendation.

This paper proposes a novel popularity-sensitive clustering method to improve the “long tail” item recommendation and makes the following two contributions.

First, we propose a clustering objective function that directly models item popularity in each cluster, and adaptively normalizes item popularity by the total item popularity associated with the item and the cluster. We design such an objective function that its optimal solution tends to have more evenly distributed item popularity in each cluster. Consequently, the less popular items in resulting clusters have a better chance to be recommended correctly.

Second, we propose an incremental optimization algorithm to optimize the proposed object function. We conduct a comprehensive experimental evaluation of the proposed clustering method on three real world datasets against three other state-of-the-art clustering methods. Our experimental results show that the proposed popularity-sensitive clustering method indeed produces clusters with more evenly distributed item popularity, and it improves not only the “long tail” recommendation accuracy, but also overall recommendation accuracy for a wide range of recommendation methods.

The rest of this paper is organized as follows. Section 2 provides some discussions on related work on the “long tail” problem and clustering collaborative filtering models. Section 3 describes the motivation of popularity-sensitive clustering. Section 4 describes the graph model of item popularities in clusters and proposes our popularity-sensitive clustering algorithm with a discussion on its objective function and optimization method. Section 5 presents a comprehensive experimental evaluation of the proposed clustering method on three real world datasets. Finally, Section 6 provides some concluding remarks.

## 2. RELATED WORK

Collaborative filtering (CF) techniques are the most widely used and well performing algorithms that have been proposed developed in the past for recommender systems [1, 8, 16]. Among the challenges that collaborative filtering techniques have to face, such as cold start, diversity, etc., the problem of "long tail" has been identified and studied to a large extent [10, 21, 13, 15, 14, 2, 22]. Recommendations from the "long tail" of the popularity distribution of items are generally considered to be particularly valuable, but the recommendation accuracy tends to decrease towards the "long tail".

The study of "long tail" came in different perspectives. [2] showed the importance of recommending "long tail" products and how this is going to change the e-commerce. [15] proposed an accuracy metric that takes the effect of "long tail" into consideration. Many recommendation methods were proposed to tackle the problem of "long tail". There are specific recommendation methods targeting only "long tail" items [10, 21], and also are ones trading-off between overall recommendation accuracy and "long tail" performance [13, 14]. For example, Eigenapp is a recommendation algorithm for recommending mobile apps in online mobile app stores proposed by [13], which employs an item normalization based on global popularity and a PCA analysis to promote less popular items. In [14], a graph model was proposed that has an explicit "long tail" factor to trade off among accuracy, diversity, and long-tail. Both [13] and [14] achieved better recommendation accuracy for "long tail" items with the sacrifice of overall recommendation accuracy. To the best of our knowledge, there has not been methods whose targets are to improve overall recommendation accuracy and "long tail" accuracy simultaneously like ours.

The technique of clustering has been used together collaborative filtering from the very beginning [12, 7]. Studies used clustering to improve efficiency of recommender systems [12], or to generate collection of user data that share similar behaviors or close relationships [4, 18, 17, 23, 20, 7, 6], based on which collaborative filtering techniques can be applied to. Some of the proposed clustering methods focused on forming clusters of users or items [12, 4, 18, 17, 23], and others focused on finding co-clusters [20, 7, 6], clusters of a subset of users and a subset of items. For example, [20] proposed a multiclass co-clustering method that allows users and items present in multiple clusters. To the best of our knowledge, there has not been clustering methods proposed to solve the problem of "long tail".

## 3. MOTIVATION

The tradeoff between item popularity and recommendation accuracy has been a well examined problem for Top-$N$ recommender systems [15, 10, 21]. Recommendations from the

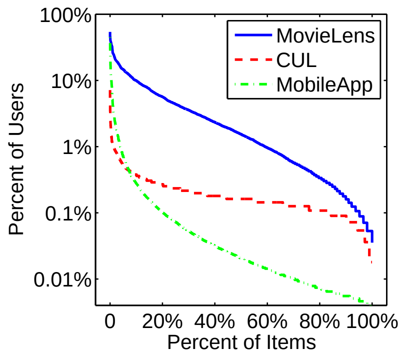

**Figure 1: Distribution of item popularities with items sorted by popularity**

"long tail" of the popularity distribution of items are generally considered to be particularly valuable, while the recommendation accuracy tends to decrease towards the "long tail". The more unbalanced a popularity distribution is, the severer the problem is. In other words, for datasets that is highly unbalanced, recommending less popular items usually means a decrease in accuracy.

In this paper, we examined three datasets **MovieLens**, **CUL**, and **MobileApp** from movie, reference papers, and mobile app domains, respectively. In Figure 3, we show the distribution of item popularites in terms of percentage of total users, with items sorted by popularity. The MobileApp dataset exhibits the "long tail" phenomenon most significantly, i.e., only a few popular items are used by the majority of users and the majority of apps are not being used in any significant sense (say being used by more than 0.1% of users). The CUL dataset exhibits the "long tail" shape as well, however, with a key difference that the majority of papers (greater than 90%) are being cited by 1% to 0.1% of users. The MovieLens dataset on the other hand is less challenging for unpopular movies, as the majority of movies has more than 1% of viewers. Recommendations from the "long tail" of the popularity distribution of items are particularly valuable for MobileApp and CUL.

We propose to tackle this problem from a clustering perspective. In addition to reaching the common goals of clustering, such as grouping users with similar behaviors or to increase the density of the user-item matrix for each cluster, our clustering method also aims to make the popularity distribution within each cluster more balanced and to reduce the effect of "long tail". In doing so, our popularity sensitive clustering method gives less popular items a better chance to be recommended correctly in resultant clusters without sacrificing the accuracy of overall recommendation.

Note that, normalizing item weights according to its global popularity in the entire dataset before performing clustering is another option to eliminate the differences between popular items and less popular items. However, normalizing item weights alone often degrades the quality of clusterings, as this approach imposes a global view of item popularity so that items in the "long tail" take higher responsibility to form **all** clusters regardless of whether they possess enough information to separate users. For example, consider the MobileApp dataset shown in Figure 3. The majority of apps are not being used in any significant sense, hence they cannot be used to separate users successfully. To overcome this limitation, our clustering method directly models the item popularity in each cluster, and prohibits all clusters being generated by less popular items alone.

## 4. POPULARITY-SENSITIVE CLUSTERING TOP-*N* RECOMMENDATION MODEL

### 4.1 Problem Formulation

Let $T$ denote the $m \times n$ user-item matrix where $m$ is the number of users and $n$ is the number of items. In this paper, we solve the problem of clustering on $T$ to produce a partition of users $(\pi_1, \ldots, \pi_k)$, where $k$ is the number of clusters and $\pi_i$ is the set of users that belong to cluster $i$. Consequently, the user-item matrix $T_i$ of cluster $i$ can be derived from $T$ by including rows of users in $\pi_i$. Our goal is to produce such a partition, so that the accuracy of collaborative filtering is improved for both popular and less popular items when recommendations are made on individual $T_i$s.

### 4.2 Normalized Popularity Graph

Our proposed clustering method is based on a local view of item popularity within each cluster, which is modeled by **Normalized Popularity Graph**.

In many top-$N$ recommendation applications, $T$ is a binary matrix, e.g., $T$ representing whether a user rated a movie or not or whether a user downloaded a mobile application or not. Therefore, we assume $T$ is binary to simplify the discussion.

*Definition 1.* **(Popularity Matrix)** Given $k$ clusters $(\pi_1, \ldots, \pi_k)$, the Popularity Matrix $P$ is a $n \times k$ item-cluster matrix, whose element $p_{ij}$ denotes the popularity of item $i$ in cluster $j$.

Given a user-item matrix $T = [t_{xy}], 1 \leq x \leq m, 1 \leq y \leq n$, and $k$ clusters $(\pi_1, \ldots, \pi_k)$, $p_{ij} = \frac{\sum_{l \in \pi_j} t_{li}}{|\pi_j|}, 1 \leq i \leq n, 1 \leq j \leq k$, where $|\pi_j|$ is the number of users in cluster $j$. Note that, the vector $(p_{1j}, p_{2j}, \ldots, p_{nj})$ is indeed the **centroid vector** $C_j$ of cluster $j$, i.e., $C_j = \frac{\sum_{l \in \pi_j} u_l}{|\pi_j|}$, where $u_l$ is the $l$th row of $T$.

*Definition 2.* **(Popularity Graph)** Given $k$ clusters $(\pi_1, \ldots, \pi_k)$, the Popularity Graph $G_P = (V, E)$ is a bipartite graph, in which two disjoint sets of vertices $V_i$ and $V_c$ correspond to items and clusters, respectively. The weight of the edge between item $i$ and cluster $j$ is defined by $p_{ij}$ in the popularity matrix $P$. Clearly, $V = V_i \cup V_c$, $|V_i| = n$, and $|V_c| = k$.

For a given popularity graph, we propose two types of normalization, namely, **item-wise (row-wise) normalization** and **cluster-wise (column-wise) normalization**. The item-wise normalization considers all edges adjacent to an item $i$, and uses the sum of their weights $\sum_{l=1}^{k} p_{il}$ as a factor to balance popular items and unpopular items.The cluster-wise normalization considers all edges adjacent to a cluster $j$, and uses the sum of their weights $\sum_{l=1}^{n} p_{lj}$ as a factor to balance dense clusters and sparse clusters. Here, we call cluster $j$ dense (or sparse), if its user-item matrix $T_j$ is dense (or sparse). Applying these two types of normalization, $p_{ij}$ is recalculated as follows:

$$p'_{ij} = \frac{p_{ij}}{\sqrt{\sum_{l=1}^{k} p_{il}} \sqrt{\sum_{l=1}^{n} p_{lj}}} \tag{1}$$

*Definition 3.* **(Normalized Popularity Graph)** Given a popularity graph $G$, the Normalized Popularity Graph $G_{NP} = (V, E)$ contains the same set of vertices and edges, and the normalized weight of the edge between item $i$ and cluster $j$ is defined by Equation 1.

Note that applying these two types of normalization is to put a popularity weight between an item and a cluster into the context of the overall popularity of an item and the overall density of a cluster. Actually, normalization based on total weights of edges adjacent to a vertex is often applied to semantic web graphs [3]. Also note that, other form of the normalization such as $L_2$-norm can also be adopted. In our preliminary study, they achieved similar performance, so we chose the simpler form for efficiency reasons.

### 4.3 Objective Function

Given a user-item matrix $T$ and normalized popularity graph $G_{NP} = (V, E)$, it is natural to define the affinity of user $i$ to cluster $j$ using the collection of normalized popularity scores $p'_{lj}$ of all items that are possessed by user $i$. That is, $\sum_{l=1}^{n} t_{il} \cdot p_{lj}$. Hence, our proposed clustering method tries to find a user partition $(\pi_1, \ldots, \pi_k)$ to maximize the following objective function:

$$Q = \sum_{l=1}^{k} \sum_{i \in \pi_l} \sum_{j=1}^{n} t_{ij} \cdot p'_{jl} \tag{2}$$

Substituting $p'$ in Equation2 with Equation1, we can obtain

$$Q = \sum_{l=1}^{k} \sum_{i \in \pi_l} \sum_{j=1}^{n} t_{ij} \cdot \frac{p_{jl}}{\sqrt{\sum p_{j,*}} \sqrt{\sum p_{*,l}}}. \tag{3}$$

We now illustrate this normalized popularity graph based objective function by the following example.

**Example 1:** Suppose we have four users and five items, and their user-item matrix is as follows:

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Now, with a partition $\pi_1 = (u_1, u_2), \pi_1 = (u_3, u_4)$, we can construct its popularity graph and normalized popularity graph as shown in Figure 2 (a) and (b), respectively. The objective function value of this partition is 4.09. The objective

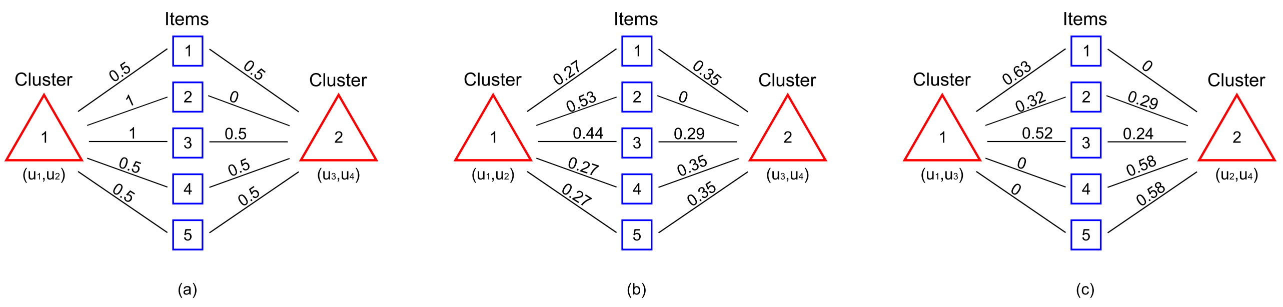

**Figure 2: For Example 1, (a) popularity graph, (b) normalized popularity graph, and (c) normalized popularity graph with the optimal partition.**

function is optimized at a value of 5.47 with the partition $\pi_1 = (u_1, u_3), \pi_2 = (u_2, u_4)$, whose normalized popularity graph is shown in Figure 2 (c).

It is clear that $\sum p_{j,*}$ and $\sum p_{*,l}$ in Equation 3 can be viewed as term weighting factors to popularity term $p_{jl}$. Conceptually, the affinity of user $i$ to cluster $l$ is determined by all items that are possessed by user $i$ collectively. With item-wise normalization, items with large total popularity scores (either because they are popular items or they become popular in some clusters) become less important, and items with small total popularity scores (either because they are unpopular items or they become unpopular in some clusters) become more important. Note that, $\sum p_{j,*}$ and $\sum p_{*,l}$ are changing as the clustering process proceeds, which makes the importance of each item self-adaptive. The clusters are formed with increasing importance of less popular items in each cluster, and decreasing importance of popular items in each cluster, which prevents popularity scores of popular items becoming too high and promotes popularity scores of unpopular items in each cluster. Consequently, the popularity scores of all none-zero items in each cluster are more evenly distributed, in other words, become less "long tail".

## 4.4 Popularity Sensitive Clustering Algorithm

We propose an incremental optimization algorithm **Popularity Sensitive Clustering (PSC)** in Algorithm 1 to optimize Equation 3.

The algorithm PSC starts with a random partition $(\pi_1, \ldots, \pi_k)$ and an objective function value associated with this partition. It then proceeds with a sequence of iterations. Within each iteration, it loops through all users and checks whether there exists a move of a user from its current cluster to a new one that can improve the objective function value. And if so, the user will be moved to the new cluster. The algorithm terminates when there is no such move in an iteration. Since the objective function value is always increased by each iteration, the algorithm will terminate eventually.

An alternative way of giving unpopular items more weights during clustering is to perform column by column normalization on the user-item matrix $T$ before invoking clustering algorithms. Column normalization is a widely used technique in recommendation methods to prevent popular items dominating recommendations [5, 13]. Here we present a Normalized K-Means algorithm in Algorithm 2, which first nor-

**Algorithm 1** Popularity-Sensitive Clustering($T, k$)

```
Require: A user-item matrix $T$, and the number of clusters $k$.
Ensure: A $k$-way partition of users $(\pi_1, \ldots, \pi_k)$.
 1: Randomly generate $(\pi_1, \ldots, \pi_k)$;
 2: Compute $Q$ according to $(\pi_1, \ldots, \pi_k)$ and $T$ by Equation 3;
 3: $ChangeMade \leftarrow True$;
 4: while $ChangeMade$ do
 5:   $ChangeMade \leftarrow False$;
 6:   for every user $i$ do
 7:     for every cluster $l$ do
 8:       Compute $Q'$ as if user $i$ is moved to cluster $l$;
 9:       if $Q' > Q$ then
10:         Update $(\pi_1, \ldots, \pi_k)$ by moving user $i$ to cluster $l$;
11:         $Q \leftarrow Q'$;
12:         $ChangeMade \leftarrow True$;
13:         Break;
14:       end if
15:     end for
16:   end for
17: end while
18: return $(\pi_1, \ldots, \pi_k)$
```

**Algorithm 2** Normalized K-Means($T, k$)

```
Require: A user-item matrix $T$, and the number of clusters $k$.
Ensure: A $k$-way partition of users $(\pi_1, \ldots, \pi_k)$.
 1: Normalize $T$ column by column;
 2: Randomly generate $(\pi_1, \ldots, \pi_k)$;
 3: Compute $C_1, \ldots, C_k$ for $(\pi_1, \ldots, \pi_k)$ ;
 4: while $NotConverged$ do
 5:   for every user $i$ do
 6:     $w_i \leftarrow argmax_{1 \le j \le k} \cos(C_j, u_i)$;
 7:   end for
 8:   Update $(\pi_1, \ldots, \pi_k)$ according to $(w_1, \ldots, w_m)$;
 9:   for every cluster $j$ do
10:     $C_j \leftarrow \frac{\sum_{l \in \pi_j} u_l}{|\pi_j|}$;
11:   end for
12: end while
13: return $(\pi_1, \ldots, \pi_k)$
```

malizes $t_{ij}$ to be $t'_{ij} = \frac{t_{ij}}{\sum_{l=1}^{m} t_{lj}}$, and then invokes K-Means to obtain a $k$-way partition $(\pi_1, \ldots, \pi_k)$.

Comparing PSC and Normalized K-Means, we see two important differences. First, PSC models item popularity within each cluster directly, whereas Normalized K-Means is based on item popularity in the entire dataset. Second, PSC forms clusters with adaptive normalization factors, whereas the importance of items is determined once and remains fixed for Normalized K-Means. Hence, PSC is more aggressive in making popularity scores of all none-zero items in each cluster evenly distributed.

## 4.5 Efficiency Analysis

The PSC algorithm normally terminates after a constant number of iterations. Within each iteration, the most expensive step is line 8 in Algorithm 1, and we need to find an efficient way to calculate $Q'$.

Given a partition $(\pi_1, \ldots, \pi_k)$ and the user-item matrix $T$, the **composite vector** $D_i$ of cluster $i$ is defined as $D_i = \sum_{l \in \pi_i} u_l$. Hence, we can rewrite the objective function in Equation 3 as follows:

$$\begin{aligned} Q &= \sum_{l=1}^{k} \sum_{i \in \pi_l} \sum_{j=1}^{n} t_{ij} \cdot \frac{p_{jl}}{\sqrt{\sum p_{j,*}}\sqrt{\sum p_{*,l}}} \\ &= \sum_{l=1}^{k} \sum_{j=1}^{n} D_{lj} \cdot \frac{p_{jl}}{\sqrt{\sum p_{j,*}}\sqrt{\sum p_{*,l}}} \end{aligned} \qquad (4)$$

Suppose user $i$ is moved from cluster $l_1$ to $l_2$. The composite vectors $D_{l_1}$ and $D_{l_2}$ are changed and can be calculated in $O(n)$ time. The popularity vectors (i.e., centroid vectors) of cluster $l_1$ to $l_2$ are also changed and can be calculated in $O(n)$ time. Given the updated popularity vectors, we can obtain the updated $\sqrt{\sum p_{j,*}}$ and $\sqrt{\sum p_{*,l}}$ in constant time. Thus, the updated $Q'$ can be calculated in $O(n + nk) = O(nk)$ time. It then follows that the overall time complexity of PSC is $O(mnk^2)$.

## 4.6 Top-$N$ Recommendation Framework

Our Top-$N$ recommendation framework is a clustering collaborative filtering model [20], where a clustering process generates clusters of data samples having similar features, and the resulting clusters are used by recommendation methods. It can be executed by the following steps.

**Step1**: Initialization: Randomly initialize the partition vector $(\pi_1, \cdots, \pi_k)$, and calculate the objective function $Q$ according to Equation 3.

**Step2**: Determining Clusters: Employ an incremental optimization procedure to optimize the objective function $Q$ by committing chancing of user cluster memberships that lead to an increment of the objective function.

**Step3**: Top-$N$ recommendation: Run any collaborative filtering method on each cluster $l$ to predict user-item scores. For each user, $N$ items with highest values are recommended.

## 5. EXPERIMENTAL EVALUATION

**Table 1: Statistics of user-item matrices for CUL, MobileApp, and MovieLens**

| Dataset | Users | Items | None-Zero Entries | Density |
|---|---|---|---|---|
| CUL | 5,551 | 16,980 | 210,537 | 0.22% |
| MobileApp | 10,000 | 4,834 | 165,032 | 0.34% |
| MovieLens | 6,040 | 3,706 | 1,000,209 | 3.70% |

We conducted our experimental evaluation on three real world datasets, and mainly evaluated recommendation accuracy of various recommendation methods using clusters produced by various clustering methods, including the proposed method PSC.

## 5.1 Datasets

The three real world datasets (**CUL** [19], **MobileApp**, and **MovieLens**) were derived from reference papers, mobile app, and movie domains, respectively.

The CiteULike dataset we used in this paper was a reference papers sharing dataset constructed from the CiteULike web site. It consists of 210,537 sharings from 5,551 users on 16,980 references. Each user has shared at least 11 references and at most 404 references and each reference paper has been shared at least by 1 user and at most by 429 users.

The MobileApp dataset we used in this paper was constructed from the server logs of Mobile Market (MM), an online mobile app store released by China Mobile, for a period of two weeks (from June 1, 2013 to June 13, 2013). Both free downloads and purchases were included in the dataset, and we used a binary presentation to denote whether a user ever downloaded a certain app. To set a minimum support for "long tail" apps, we removed apps with less than 6 users. Finally, we randomly selected 10,000 users who downloaded at least 10 apps and at most 100 apps in this two-week period to be included in the MobileApp dataset.

The MovieLens dataset we used in this paper is MovieLens-1M, a classic movie rating dataset constructed from the MovieLens web site [1]. It consists of 1,000,209 ratings from 6,040 users on 3,706 movies. For our recommendation purpose, we do not use the actual ratings of the dataset, instead, we make each entry in the user-item matrix binary with 1 indicating a user has rated a movie and 0 otherwise.

The statistics of user-item matrices for the CUL [19], MobileApp, and MovieLens datasets are shown in Table 1. From Table 1 we can see that both CUL and MobileApp are very sparse. Recall that these two datasets also have "long tail".

## 5.2 Methodology

In order to evaluate the recommendation accuracy of our proposed popularity-sensitive clustering (PSC) top-$N$ recommendation framework, we investigate four recommendation methods including two memory based algorithms of Item-based [11, 9] and User-based [11], one matrix factorization method of PureSVD [5], and recommendation by global popularity (POP). We firstly run PSC to partition the users,

[1]http://www.grouplens.org

and then run various methods to generate a recommendation list for each user. The setup of each recommendation method is listed below.

**Item-based**: In our Item-based algorithm, the similarity between two items is measured by vector cosine similarity. For each item candidate $i$, instead of summing over all items used by user $u$, we only consider the $l$ nearest items according to [13]. In our experiments, we set $l = 10$. The Item-based model can be found in [8].

**User-based**: In our User-based algorithm, the similarity between two users is measured by vector cosine similarity. The User-based model can be found in [8].

**PureSVD**: Factorize user-item matrix $T$ via SVD with $k$ features $\hat{T} = U \cdot \Sigma \cdot V$. The prediction score of user $u$ to item $j$ is $\hat{t}_{ui} = t_u \cdot V \cdot v_i^T$. In our experiments, we set $k = 16$. The PureSVD model can be found in [5].

**POP**: In this paper, the popularity score for each item is defined as the number of usages. For each user, we discard the items that the user has used before and recommend top-$N$ items in the remaining list.

We compare our proposed popularity-sensitive clustering method with one baseline method, two state-of-the-art clustering methods for collaborative filtering, and the normalized K-Means method proposed in section 4 as well. The baseline method, denoted as "base", is simply having all users in a single cluster. We can see the original relative performance of various recommendation methods from this method. The two state-of-the-art clustering methods are K-Means [17], and MCoC, a multiclass co-clustering method proposed by [20].

We employ the same top-$N$ recommendation framework with these five clustering methods in the first step, and determine which clustering method is most effective in terms of improving recommendation accuracy of various recommendation methods. We run each clustering method 10 times with random initial partition, and the average recommendation accuracy over 10 runs is calculated and reported. The MCoC method needs parameter tuning. For MovieLens, we use the same set of parameters reported in [20] and manually tune parameters with the best performance for other datasets. All experiments are conducted on a machine with a 2.67Ghz Quad-core Intel Core 2 processor and 8GB memory.

## 5.3 Evaluation Metric

All three datasets were separated into a training set and a testing set. The various clustering collaborative filtering models recommend a list of items to each user based on the training set, and the list was evaluated against the testing set.

For the MobileApp dataset, each download is associated with a time stamp, which allows us to prepare the training and testing sets for the case of recommending next downloads. In particular, we used the very last download in the two-week period of each user as the testing set, and the rest of downloads were included in the training set. For the MovieLens and CUL datasets, to construct evaluation data, we randomly selected one item per user, and the remaining data made up the training set.

We evaluated the performance of clustering collaborative filtering model in precision and recall. When $N$ items are recommended to each user, precision and recall for all $m$ users are computed by:

$$precision@N = N_{hits}/(N \times m)$$

$$recall@N = N_{hits}/N_{total}$$

where $N_{hits}$ is the number of hits in the top-$N$ list for all users, $m$ is the number of users, and $N_{total}$ is the total number of downloads in the testing set. Obviously, in our evaluation, $N_{total}$ is equal to $m$, which means $recall@N = N \times precision@N$, so we only reported recall@N in this paper.

Since the goal of our proposed clustering method is to improve the recommendation accuracy not only for popular items, but also for less popular items, we used two types of recall@N, one for evaluating against all items in the testing set, the other for evaluating against the remaining items in the testing set after eliminating the 100 most popular items in the training set.

## 5.4 Experimental Results

The first set of experiments were designed to verify whether our clustering method can improve the popularity distribution in resultant clusters. The "long tail" distribution is characterized by uneven distribution of popular items and unpopular items. It is difficult to compare directly the popularity distribution of items in resultant clusters of two clustering methods, as there is no one to one mapping of clusters in the two clustering results. Instead, we looked at the item directly and calculated the average popularity of each item among clusters that have non-zero downloads of that item. That is, given a clustering result $(\pi_1, \cdots, \pi_k)$, for each item $i$, we calculated the average item popularity among clusters as $average(p_{il}), 1 \leq l \leq k$ and $p_{il} \neq 0$. We applied K-Means, Normalized K-Means, and PSC to produce 25 clusters on all three datasets, and plotted the average item popularity distributions in Figure 3. We also plotted the global popularity of each item to serve as a baseline.

First of all, we see a trend of increasing item popularity in clusters produced by K-Means and NK-Means over item global popularity for all items, which indicates the resultant clusters indeed correspond more dense regions. The increase of unpopular items is more obvious for all clustering methods on all datasets, whereas the increase of popular items is not always so. For CUL, the global popularity of popular items is relatively low, in which case the improvement is very clear. On the other hand, our proposed method PSC tries to balance the presence of popular items and unpopular items. For the two datasets with "long tail", CUL and MobileApp, it is very clear that PSC is the one among all clustering methods that gives the highest average popularity in clusters to items that are less popular (not 5% most popular items). Actually, PSC even decreases the item popularity in clusters for the most popular items on both MobileApp and MovieLens. Hence, PSC indeed achieves the goal of making

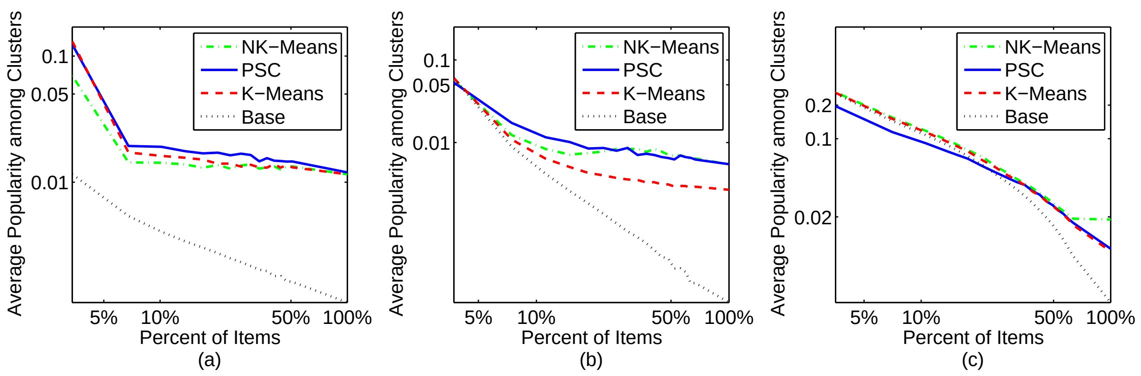

**Figure 3: Distribution of average item popularities among clusters with items sorted by global popularity for (a) CUL, (b) MobileApp, and (c) MovieLens.**

item popularity in clusters distributed more evenly and less "long tail", which potentially can give less popular items a better chance of being recommended correctly.

The second set of experiments were designed to verify whether our proposed clustering method can improve the recommendation accuracy in resulting clusters. We used PSC to produce 5, 15, and 25 clusters and employed various recommendation methods on each cluster to produce recommendation list. The results of averaged recall@10 over 10 runs by PSC, as well as other clustering methods are shown in Table 2, Table 3, and Table 4. The column labeled "Base" corresponds to the case where no clustering method is used, and recommendations are made based on the original user-item matrices. The best recall@10 values for each row are in bold face, and the overall best value for each dataset is underlined as well.

There are several trends we can observe from Table 2, Table 3, and Table 4. First of all, for most of the cases our proposed method PSC leads to the best recall and the best recall without the 100 most popular items. Actually, PSC is the only method that can improve the recommendation accuracy consistently for all recommendation methods on all datasets, and it improves both the overall recommendation accuracy and the "long tail" recommendation accuracy. MCoC has a better performance for recommending less popular items on MobileApp, however its recall values for all items are even worse than the baseline for all recommendation method on MobileApp. PSC outperforms the other three clustering methods significantly on CUL and MobileApp. Even for MovieLens, not a typical dataset with a "long tail", PSC achieves the best performance for less popular items, and slightly better performance in terms of the overall recall than K-Means.

In terms of recommendation methods, all of them can achieve better recommendation accuracy with the help of clustering methods. The improvements in terms of overall recall@10 vary from method to method on various datasets. On CUL, the improvement by PSC for PureSVD and POP is more significant, with a factor of 2 and 3 times in terms of recall@10, and the improvement by PSC for User-based and item-based is about 20% and 5%. On MobileApp, the improvement by PSC is between 5% to 10% for all methods. On MovieLens, the improvement by PSC for PureSVD is about 6%, and for other three methods is larger than 36%. The improvements in terms of recall@10 without the 100 most popular items are even more significant for all recommendation methods on all datasets. Note that the recommendation method that achieves the best recommendation accuracy varies for different datasets, namely, User-based on CUL, Item-based on MobileApp, and PureSVD on MovieLens.

In terms of other clustering methods, K-Means is the next best performed one. On MobileApp and MovieLens, it can improve the overall recommendation accuracy for most of the recommendation methods. The Normalized K-Means only performs well on CUL, which indicates that simply normalizing items according to its global popularity does not work well in general.

Using the best performed recommendation method on each dataset as an example, namely, User-based on CUL, Item-based on MobileApp, and PureSVD on MovieLens, we further plotted the recall curves by varying $N$ from 10 to 50. The curves for overall recall and "long tail" recall obtained with 25 clusters generated by various clustering methods are shown in Figure 4 and Figure 5, respectively.

From Figure 4 and Figure 5, we can see that PSC consistently leads to the best performance on each dataset with the best performed recommendation method. On CUL and MobileApp, the two sparse datasets with "long tail", the curves show the superior performance of PSC on "long tail" items as $N$ grows. The performance of other clustering methods, on the other hand, is not stable to a large extent. For example, K-Means is the worst one on CUL, but the second best one on MobileApp. Normalized K-Means is the second best one on CUL, but the worst one on MovieLens. MCoC improves the recommendation accuracy for "long tail" items consistently on all three datasets, however, it cannot improve the overall recommendation accuracy at the same time.

Finally, we evaluated various clustering methods by their efficiency, and showed their runtime in minutes on all three datasets in Table 5, Table 6, and Table 7. We did not show the runtime of Normalized K-Means, as its runtime is very similar to the one of K-Means.

**Table 2: Recommendation Accuracy on CUL in terms of (a) Recall@10 and (b) Recall@10 without the 100 most popular items**

(a) Recall@10

| | | MCoC | | | PSC | | | K-Means | | | NK-Means | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Methods | Base | 5 gs | 15 gs | 25 gs | 5 gs | 15 gs | 25 gs | 5 gs | 15 gs | 25 gs | 5 gs | 15 gs | 25 gs |
| Item-based | 0.177 | 0.165 | 0.168 | 0.169 | **0.187** | 0.178 | 0.182 | 0.147 | 0.126 | 0.123 | 0.157 | 0.149 | 0.150 |
| User-based | 0.170 | 0.162 | 0.165 | 0.169 | 0.188 | 0.202 | **0.204** | 0.156 | 0.148 | 0.145 | 0.177 | 0.190 | 0.193 |
| PureSVD | 0.051 | 0.071 | 0.073 | 0.080 | 0.096 | 0.135 | **0.158** | 0.079 | 0.092 | 0.117 | 0.086 | 0.091 | 0.104 |
| POP | 0.035 | 0.025 | 0.032 | 0.022 | 0.038 | 0.047 | **0.062** | 0.027 | 0.040 | 0.042 | 0.036 | 0.040 | 0.043 |

(b) Recall@10 without the 100 most popular items

| | | MCoC | | | PSC | | | K-Means | | | NK-Means | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Methods | Base | 5 gs | 15 gs | 25 gs | 5 gs | 15 gs | 25 gs | 5 gs | 15 gs | 25 gs | 5 gs | 15 gs | 25 gs |
| Item-based | 0.185 | 0.184 | 0.177 | 0.182 | **0.193** | 0.185 | 0.188 | 0.156 | 0.131 | 0.129 | 0.166 | 0.158 | 0.160 |
| User-based | 0.148 | 0.164 | 0.158 | 0.164 | 0.176 | 0.196 | **0.199** | 0.142 | 0.137 | 0.136 | 0.161 | 0.180 | 0.185 |
| PureSVD | 0.024 | 0.050 | 0.060 | 0.068 | 0.079 | 0.125 | **0.151** | 0.064 | 0.080 | 0.108 | 0.075 | 0.076 | 0.090 |

**Table 3: Recommendation Accuracy on MobileApp in terms of (a) Recall@10 and (b) Recall@10 without the 100 most popular items**

(a) Recall@10

| | | MCoC | | | PSC | | | K-Means | | | NK-Means | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Methods | Base | 5 gs | 15 gs | 25 gs | 5 gs | 15 gs | 25 gs | 5 gs | 15 gs | 25 gs | 5 gs | 15 gs | 25 gs |
| Item-based | 0.200 | 0.200 | 0.175 | 0.175 | 0.205 | 0.211 | **0.213** | 0.204 | 0.202 | 0.202 | 0.201 | 0.195 | 0.195 |
| User-based | 0.186 | 0.181 | 0.176 | 0.172 | 0.197 | 0.201 | **0.202** | 0.189 | 0.185 | 0.184 | 0.188 | 0.189 | 0.188 |
| PureSVD | 0.179 | 0.170 | 0.146 | 0.143 | 0.185 | 0.190 | **0.192** | 0.173 | 0.176 | 0.179 | 0.145 | 0.144 | 0.145 |
| POP | 0.178 | 0.180 | 0.155 | 0.156 | 0.186 | 0.192 | **0.193** | 0.182 | 0.184 | 0.185 | 0.180 | 0.179 | 0.180 |

(b) Recall@10 without the 100 most popular items

| | | MCoC | | | PSC | | | K-Means | | | NK-Means | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Methods | Base | 5 gs | 15 gs | 25 gs | 5 gs | 15 gs | 25 gs | 5 gs | 15 gs | 25 gs | 5 gs | 15 gs | 25 gs |
| Item-based | 0.008 | 0.008 | **0.030** | 0.022 | 0.011 | 0.017 | 0.021 | 0.010 | 0.014 | 0.017 | 0.011 | 0.013 | 0.014 |
| User-based | 0.000 | 0.000 | **0.028** | 0.024 | 0.003 | 0.010 | 0.016 | 0.001 | 0.003 | 0.010 | 0.001 | 0.001 | 0.002 |
| PureSVD | 0.002 | 0.011 | 0.026 | 0.018 | 0.010 | 0.027 | **0.031** | 0.015 | 0.021 | 0.024 | 0.006 | 0.007 | 0.008 |

**Table 4: Recommendation Accuracy on MovieLens in terms of (a) Recall@10 and (b) Recall@10 without the 100 most popular items**

(a) Recall@10

| | | MCoC | | | PSC | | | K-Means | | | NK-Means | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Methods | Base | 5 gs | 15 gs | 25 gs | 5 gs | 15 gs | 25 gs | 5 gs | 15 gs | 25 gs | 5 gs | 15 gs | 25 gs |
| Item-based | 0.126 | 0.104 | 0.119 | 0.112 | 0.148 | 0.165 | **0.172** | 0.145 | 0.158 | 0.168 | 0.131 | 0.125 | 0.126 |
| User-based | 0.113 | 0.115 | 0.114 | 0.109 | 0.136 | 0.158 | **0.168** | 0.135 | 0.156 | 0.163 | 0.109 | 0.109 | 0.111 |
| PureSVD | 0.199 | 0.210 | 0.190 | 0.178 | 0.210 | **0.213** | 0.208 | 0.208 | 0.209 | 0.199 | 0.198 | 0.197 | 0.192 |
| POP | 0.084 | 0.085 | 0.086 | 0.095 | 0.121 | 0.144 | **0.157** | 0.117 | 0.143 | 0.154 | 0.082 | 0.086 | 0.085 |

(b) Recall@10 without the 100 most popular items

| | | MCoC | | | PSC | | | K-Means | | | NK-Means | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Methods | Base | 5 gs | 15 gs | 25 gs | 5 gs | 15 gs | 25 gs | 5 gs | 15 gs | 25 gs | 5 gs | 15 gs | 25 gs |
| Item-based | 0.022 | 0.017 | 0.040 | 0.042 | 0.046 | 0.073 | **0.084** | 0.041 | 0.064 | 0.075 | 0.026 | 0.027 | 0.029 |
| User-based | 0.000 | 0.004 | 0.024 | 0.031 | 0.028 | 0.065 | **0.077** | 0.027 | 0.058 | 0.068 | 0.004 | 0.005 | 0.006 |
| PureSVD | 0.086 | 0.110 | **0.138** | 0.137 | 0.118 | 0.134 | 0.134 | 0.117 | 0.129 | 0.128 | 0.093 | 0.088 | 0.082 |

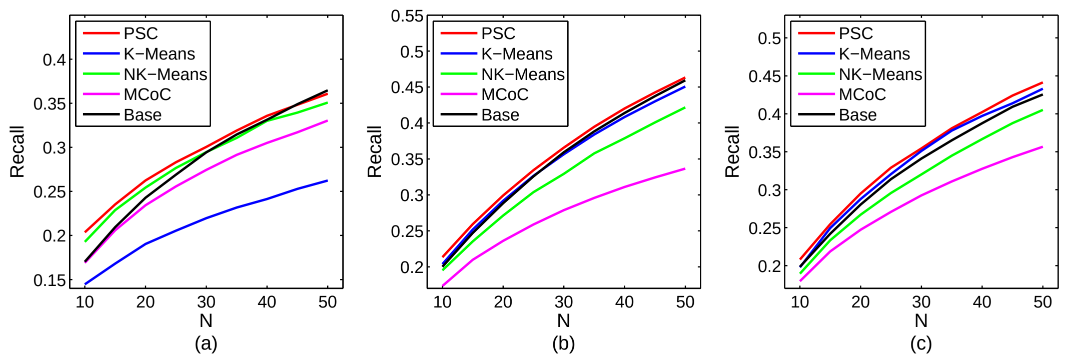

**Figure 4: Recall@N curves for (a) CUL, (b) MobileApp, and (c) MovieLens.**

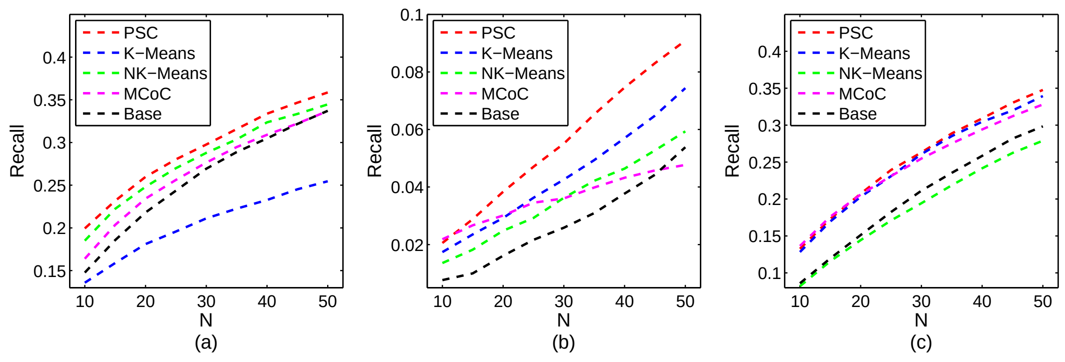

**Figure 5: Recall@N curves after removing the 100 most popular items for (a) CUL, (b) MobileApp, and (c) MovieLens.**

**Table 5: Efficiency on CUL**

| Runtime (min) | 5gs | 15gs | 25gs |
|---|---|---|---|
| K-Means | 0.017 | 0.050 | 0.067 |
| MCoC | 30.517 | 30.767 | 31.267 |
| PSC | 0.667 | 24.867 | 48.450 |

**Table 6: Efficiency on MobileApp**

| Runtime (min) | 5gs | 15gs | 25gs |
|---|---|---|---|
| K-Means | 0.017 | 0.033 | 0.183 |
| MCoC | 9.550 | 9.767 | 9.833 |
| PSC | 2.133 | 20.767 | 73.900 |

**Table 7: Efficiency on MovieLens**

| Runtime (min) | 5gs | 15gs | 25gs |
|---|---|---|---|
| K-Means | 0.017 | 0.233 | 0.400 |
| MCoC | 11.400 | 11.233 | 11.567 |
| PSC | 1.100 | 14.650 | 46.067 |

The K-Means method is the most efficient one, as it benefits from a batch optimization procedure. Since the time complexity of K-Means is linear with respect to the number of clusters, we see the runtime grows at $k$ grows from 5 to 25. The MCoC method has two steps, i.e., performing a single value decomposition in the first step, and then performing a K-Means clustering in the second step. Thus, the time for the first step dominates the overall runtime. Our proposed method PSC is more time consuming than other methods, as it incrementally evaluates a complicated objective function. It has a time complexity of $O(mnk^2)$, where $k$ is the number of clusters. So when $k \geq 15$, PSC is the most expensive one among all three clustering methods.

## 6. CONCLUSIONS

Recommendation methods usually can perform better under the help of clustering. In this paper, we propose a novel clustering model PSC based on popularity distribution to further improve the performance. Experimental results show that Popularity-sensitive clustering is a stable and outstanding model, which improves both the overall accuracy and the “long tail” accuracy to a great extent, and outperforms many existing clustering models on many recommendation methods on different datasets. Future work on efficiency is needed. We are hoping to work out a batch optimization version of PSC.

## 7. ACKNOWLEDGEMENT

This work was supported by the Ministry of Education & China Mobile Research Foundation (no.MCM20123021), the Scientific and Technical Supporting Program (no. 2012BAH-08B02), and 863 Program of China (no. 2012AA012600).

## 8. REFERENCES

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on Knowledge and data engineering*, 17:734–749, 2005.

[2] C. Anderson. *The Long Tail: Why the Future of Business in Selling Less of More*. Hyperion, 2006.

[3] B. Bamba and S. Mukherjea. Utilizing resource importance for ranking semantic web query results. In *Proc. of the second international workshop on semantic web and data mining*, 2004.

[4] M. Connor and J. Herlocker. Clustering items for collaborative filtering. In *Proc. of ACM SIGIR Workshop on recommendation systems*, 1999.

[5] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendations. In *Proc. of the fourth ACM conference on recommender systems*, 2010.

[6] T. George. A scalable collaborative filtering framework based on co-clustering. In *Fifth IEEE International Conference on Data Mining*, pages 625–628, 2005.

[7] T. Hofmann and J. Puzicha. Latent class models for collaborative filtering. In *IProc. of the 6th international joint conference on artificial intelligence*, pages 688–693, 1999.

[8] Z. Huang, D. Zeng, and H. Chen. A comparison of collaborative-filtering recommendation algorithms for e-commerce. *IEEE Intelligent System*, 22(5):68–78, 2007.

[9] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7:76–80, 2003.

[10] Y. J. Park and A. Tuzhilin. The long tail of recommender systems and how to leverage it. In *Proc. of the 2nd ACM conference on recommender systems*, 2008.

[11] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proc. of the 10th international conference on World Wide Web*, 2001.

[12] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering. In *Proc. of the 5th international conference on computer and information technology*, 2002.

[13] K. Shi and K. Ali. Getjar mobile application recommendations with very sparse datasets. In *Proc. of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2012.

[14] L. Shi. Trading-off among accuracy, similarity, diversity and long-tail: A graph-based recommendation approach. In *Proc. of the 7th ACM conference on recommender systems*, 2013.

[15] H. Steck. Item popularity and recommendation accuracy. In *Proc. of the 5th ACM conference on recommender systems*, 2011.

[16] X. Su and T. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 2009.

[17] C. Tsai and C. Hung. Cluster ensembles in collaborative filtering recommendatione. *Applied Soft Computing*, 12:1417–1425, 2012.

[18] L. Ungar and D. Foster. Clustering methods for collaborative filtering. In *Proc. of AAAI Workshop on recommendation systems*, 1998.

[19] C. Wang and D.M.Blei. Collaborative topic modeling for recommending scientific articles. In *Proc. of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2011.

[20] B. Xu, J. Bu, C. Chen, and D. Cai. An exploration of improving collaborative recommender systems via user-item subgroups. In *Proc. of the 21st international conference on World Wide Web*, 2012.

[21] H. Yin, B. Cui, J. Li, J. Yao, and C. Chen. Challenging the long tail recommendation. In *Proc. of VLDB 2012*, 2012.

[22] M. Zhang and N. Hurley. Avoiding monotony: improving the diversity of recommendation lists. In *Proc. of the 2nd ACM conference on recommender systemsg*, 2008.

[23] Y. Zhang, M. Zhang, Y. Liu, and S. Ma. Improve collaborative filtering through bordered block diagonal form matrices. In *Proc. of the 36th international ACM SIGIR conference on research and development in information retrieval*, 2013.