The Long Tail of Recommender Systems and How to Leverage It

Yoon-Joo Park
Stern School of Business, New York University
ypark@stern.nyu.edu

Alexander Tuzhilin
Stern School of Business, New York University
atuzhili@stern.nyu.edu

ABSTRACT

The paper studies the Long Tail problem of recommender systems when many items in the Long Tail have only few ratings, thus making it hard to use them in recommender systems. The approach presented in the paper splits the whole itemset into the head and the tail parts and clusters only the tail items. Then recommendations for the tail items are based on the ratings in these clusters and for the head items on the ratings of individual items. If such partition and clustering are done properly, we show that this reduces the recommendation error rates for the tail items, while maintaining reasonable computational performance.

Categories and Subject Descriptors
H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval.
I.2.6 [Artificial Intelligence]: Learning.

General Terms: Algorithm, Performance, Experimentation.

Keywords: Long Tail, clustering, recommendation, data mining

1. INTRODUCTION

Many recommender systems ignore unpopular or newly introduced items having only few ratings and focus only on those items having enough ratings to be of real use in the recommendation algorithms. Alternatively, such unpopular or newly introduced items can remain in the system but would require special handling using various cold start methods, such as the ones described in [1].

Using the terminology introduced in [2], these unpopular or new items belong to the Long Tail of the item distribution, as shown in Figure 1 for the MovieLens dataset. Following the spirit of extensive research on the Long Tail phenomena [2], these types of items should not be discarded or ignored but gainfully utilized in recommendation methods. In this paper, we study the Long Tail of recommender systems and propose a new method of managing such items from the Long Tail. In particular, we propose to split items into the head and tail parts and group items in the tail part using certain clustering methods. We show that such splitting and grouping improves recommendation performance as compared to some of the alternative non-grouping and fully-grouped methods. We demonstrate this performance improvement by running various experiments on two "real-world" datasets. Finally, we examine head/tail splitting strategies reducing error rates of recommendations and demonstrate that this partitioning often outperforms clustering of the whole itemset.

The Long Tail problem in the context of recommender systems has been addressed previously in [3] and [4]. In particular, [3] analyzed the impact of recommender systems on sales concentration and developed an analytical model of consumer purchases that follow product recommendations provided by a recommender system. The recommender system follows a popularity rule, recommending the bestselling products to all consumers, and they show that the process tends to increase the concentration of sales. As a result, the treatment is somewhat akin to providing product popularity information. The model in [3] does not account for consumer preferences and their incentives to follow recommendations or not. Also [3] studied the effects of recommender systems on sales concentration and did not address the problem of improving recommendations for the items in the Long Tail, which constitutes the focus of this paper. In [4], a related question has been studied: to which extent recommender systems account for an increase in the Long Tail of the sales distribution. [4] shows that recommender systems increase firm's profits and affect sales concentration.

Another related problem is the problem of the cold start [1]. This is the case because our approach can be viewed as a solution to the cold start problem for the items in the Long Tail that have only very few ratings. A popular solution to the cold start problem utilizes content-based methods when two items with no or only few ratings are inferred to be similar based on their content [1]. In our work, we use grouping of items in the long tail, rather than the content-based methods to identify similar items and to leverage their combined ratings to provide better recommendations.

Our work is also related to the clustering methods used in recommender systems. In particular, [9] clusters similar users into the same cluster to overcome the data sparsity problem for collaborative filtering. Also in [8], item clustering is used to improve the prediction accuracy of collaborative filtering where items were divided into smaller groups, and existing CF algorithms were applied to each group category separately. We use related clustering ideas but in the context of the Long Tail phenomenon to leverage few ratings of the items in the Long Tail.

2. BACKGROUND

In this section, we provide some background information about the Long Tail problem in recommender systems and its solutions.

2.1 Preliminaries

We assume that there is a set of items I, a set of customers C and the set of known ratings $R = \{r_{ij}\}$ provided by the customers in C

for the items in I. Let $R_i = \{r_{ij}\}$ be the set of ratings provided by customers in C for item i. We order all the items in I according to the number of ratings $|R_i|$ provided by customers for that item. The histogram in Figure 1 presents the frequency distribution of the item's rating numbers and follows the Long Tail [2] for the MovieLens dataset [5] described in Section 4. The whole itemset I can be partitioned into the head H and the tail T parts by selecting a *cutting point* α along the x-axis of the histogram.

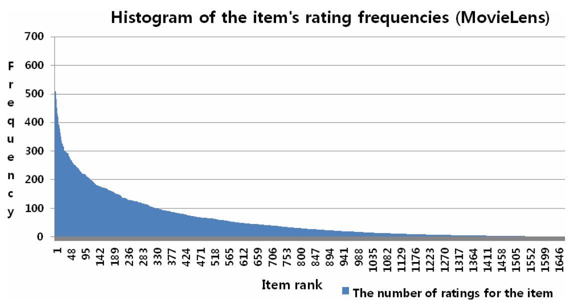


Figure 1. Histogram of the items' (movies') rating frequencies for the MovieLens data.

In this paper, we assume that the recommendations are provided as follows. First, we group items in I according to some clustering algorithm [7]. Then for each cluster of items I_k we build a data mining model predicting unknown ratings in the cluster I_k based on the known ratings $R_k = \{r_{ij}\}_{i \in Ik}$ and the parameters of items in cluster I_k and customers in C. For example, we can build a linear regression model for a cluster of items I_k using the known ratings for that cluster to estimate the unknown ratings for the items in I_k. We can also determine the error rates of these models, such as RMSE and MAE, by testing performance of these data mining models on the holdout data.

In order to build these data mining models, we first need to specify the variables pertaining to items I and customers C. We assume that customers C have certain attributes, such as name, age, gender and address, associated with the customers and that items in I have attributes, such as item name, price, size and weight, associated with them. In addition to these attributes, we also introduce derived variables *DV* for the customers and items that are computed from the customer and item attributes and the ratings information. Some examples of derived variables are:

1) the average rating provided by a customer for the rated items
2) the total number of ratings that the customer provided
3) the total number of ratings for an item.

The independent variables used in the aforementioned data mining models include the item-related and the derived variables.

If we do not group items, as explained above, and build data mining models for *each* individual item i in I to estimate unspecified ratings for i, we call it *Each Item* (*EI*) recommendation method. For example, in case of the MovieLens dataset, we ordered 1682 movies from that dataset based on the number of available ratings for a movie. Than we build a predictive model for each of the 1682 movies using the ratings of that particular movie (thus, we built 1682 models in total). The independent variables are some of the derived variables, such as the number of ratings that the customer provided, the average popularity of the movies that customer gives high rating and the popularity of the movie, etc. For example, if movie *Toy Story* had 272 ratings, we can build a linear regression model to predict the unknown ratings for that movie, use RMSE to measure performance of the model, and apply 10-fold cross validation to compute RMSE for that movie. This process was repeated 1682 times for each movie in MovieLens. As Figure 1 demonstrates, the movies in its long tail have only few ratings, and predictive models are learned from only few training examples using the EI method. Finally, since we used 10-fold cross validation, the minimal number of ratings needed for the model-building purposes was 10, which was the case with MovieLens.

In contrast, when we group items by applying clustering methods to the whole itemset I, we call it *Total Clustering* (*TC*) recommendation method. In other words, TC clusters the whole itemset I into different groups and builds rating predictive models for each resulting group. Finally, if we split itemset I into the head H and tail T, apply clustering only to the tail T while leaving head items un-clustered, and build data mining models for each cluster in tail T and individual models in head H, we call it *Clustered Tail* (*CT*) recommendation method.

The main problem with the *Each Item* (*EI*) recommendation method is that only few ratings are available in the Long Tail, and the performance rates of these models deteriorate in the Long Tail of the itemset I. We describe this problem in Section 2.2 and present various ways to address it in the rest of the paper.

2.2 The Long Tail Problem of Recommender Systems

We used the *Each Item* (*EI*) method to build rating estimation models for individual items in I, as described in Section 2.1. We have used Weka [7] to repeat this model building process across two datasets (MovieLens and BookCrossing), two types of performance measures (MAE and RMSE) and nine types of predictive models: (1) SimpleLinearRegression (*SLR*), (2) Gaussian radial basis function network (*RBF*), (3) Support vector machines (*SVM*), (4) K-nearest neighbours (*KNN*), (5) Locally-weighted learning (*LWL*), (6) Bagging classifier (*Bagging*), (7) DecisionStump tree (*DecisionStump*), (8) M5P model tree (*M5P*) and (9) 0-R Rule Induction classifier (*ZeroR*). Furthermore, we have build these individual item models using five sets of derived variables that served as independent variables in the model. Therefore, the total number of experiments for *Each Item* method are 90 (2 × 9 × 5) for each data set.

The performance results for some of these experiments are presented in Figures 2 and 3. Figures 2(a, b) show the MAE and RMSE error rates respectively for each of the aforementioned nine predictive models for the MovieLens dataset. The movies in the graphs are ordered according to the number of ratings (ranks) that are plotted on the x-axis. Figures 3(a, b) provide the same measurements information, but for the BookCrossings dataset.

All the four figures Fig 2-3(a, b) show that the error rates increase in the tail of the figures (for the items with only few ratings). To demonstrate this effect more clearly, we performed the correlation analysis and computed Pearson's correlation coefficients between the rating numbers and error rates. The results are presented in Table 1, and it shows that all the 90 models for the BookCrossing dataset have significant negative correlations between the rating numbers and the error rates and 70 (out of 90) models have significant negative correlation for the MovieLens dataset. This result demonstrates that, when the rating numbers decrease, then the error rates tend to increase.

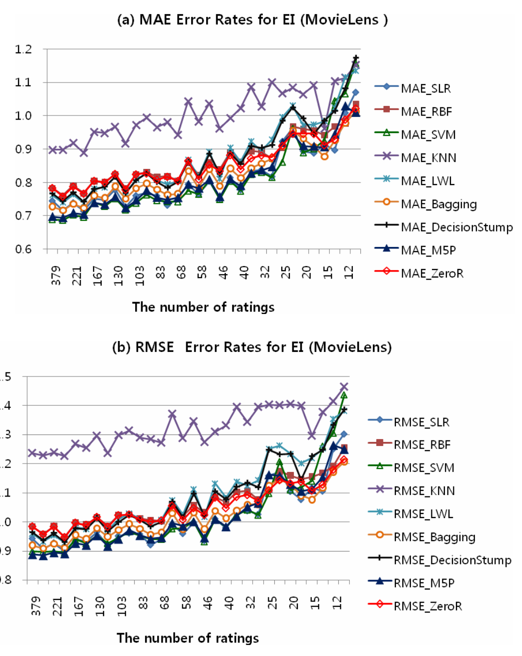


Figure 2. Error rates for *Each Item* method for the MovieLens dataset for different predictive models. On the x-axis is the number of ratings for movies in decreasing order.

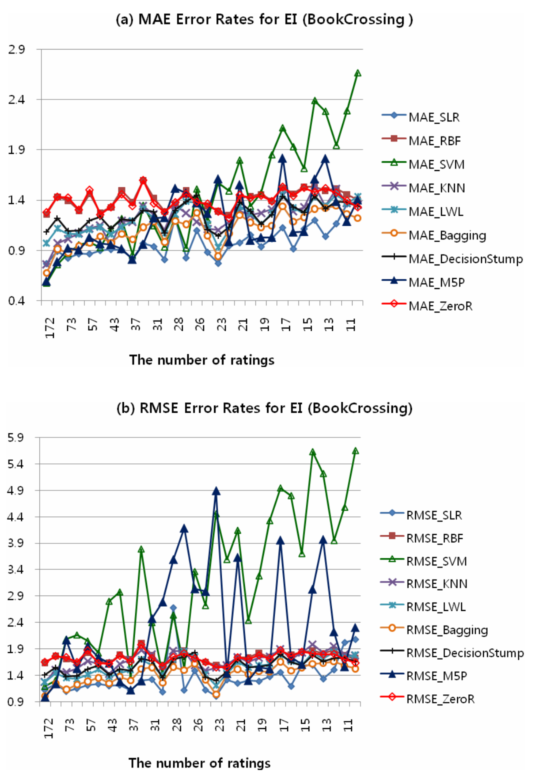


Figure 3. Error rates for *Each Item* method for the BookCrossing dataset for different predictive models. On the x-axis is the number of ratings for books in decreasing order.

In summary, we showed that for the EI rating estimation method, the error rates tend to increase for the low-ranked items in the tail of itemset *I*. We call this problem the *Long Tail Recommendation Problem (LTRP)*. It occurs because rating prediction models do not have enough data for the less popular items in the Long Tail. We address this problem in the rest of this paper. One way to do it is to cluster items in *I* so that predictive models are learned using more data, thus, decreasing error rates for the less popular items. We describe this approach next.

Table 1. Correlation analysis result for the graphs in Figures 2 and 3 for the BookCrossing and MovieLens datasets

Pearson's Correlation Coefficient	# of *EI* models for BookCrossing	# of *EI* models for MovieLens
Significant Correlation	90 (/90)	70(/90)
Less than -0.5	70	43
Less than -0.6	66	22
Less than -0.7	40	5

2.3 Solution to the Long Tail Problem: Clustering the Whole Itemset

Since the LTRP problem is caused by the lack of data to build good predictive models in the tail, clustering items using the Total Clustering (*TC*) method from Section 2.1 can be a reasonable solution since it can provide more data for the less popular items.

In this section we compare the *EI* and the *TC* methods. The experimental settings for the *TC* method are the same as for *EI*, as described in Section 2.2; however the *TC* method has an additional factor – the number of clusters. Thus, for the *TC* models, we consider nine data mining methods, five sets of the independent variables, two performance measurements and five clustering groups having 10, 20, 30, 40 and 50 clusters in total. We cluster items in these groups using the Expectation-Maximization (*EM*) clustering method [7]. Thus, the number of experiments for the *TC* case becomes 450.

For example, in case of the MovieLens dataset, we cluster 1682 movies into 10 group using the *EM* method [7] and build a predictive model, e.g., a Support Vector Machine for each group (10 SVM models in total). If we want to predict the unknown rating of movie *The Other Boleyn Girl* for customer *C* then the *TC* method, first, determines into which of these 10 groups that movie belongs. If the movie belongs to group *G5*, consisting of 30 other movies having 10000 transactions among them, then *TC* applies SVM method to group *G5* and computes RMSE error rates using 10-fold cross validation on these 10000 ratings. This process was repeated 10 times on MovieLens for each cluster.

Figures 4 and 5 show the RMSE rates for the *TC* and *EI* methods across the MovieLens and BookCrossing datasets respectively. Figure 4 uses the Simple Linear Regression (*SLR*) method and 10 clusters for the *TC* method. Figure 5 uses the Locally-Weighted Learning (*LWL*) method and also 10 clusters for the *TC* method. These two graphs clearly show that the *TC* outperforms the *EI* method, especially in the tail of the distribution, where the gap between the two lines is clearly visible. In order to formally verify this visual observation, we performed the paired t-test. Table 2 presents the paired t-test results and shows that for the MovieLens data in 448 (out of 450) cases the error rates of the *TC* models are significantly smaller than the *EI* error rates at the 95% confidence levels. Likewise, in

440 (out of 450) cases the error rates of the *TC* models are significantly smaller than for the *EI* method for BookCrossing data. Thus, we conclude that the *TC* outperforms the *EI* method.

Table 2. Paired t-test result comparing *Total Clustering (TC)* vs. *Each Item (EI)* methods.

Dataset	$Err._{CT}$<$Err._{EI}$	$Err._{EI}$<$Err._{CT}$	$Err._{CT}$=$Err._{EI}$
MovieLens	448	0	2
BookCrossing	440	1	9

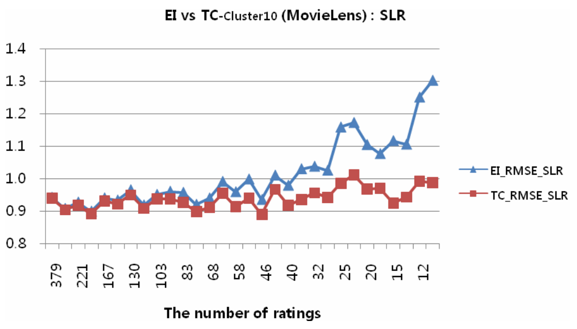


Figure 4. RMSE error rates of *Total Clustering* (with 10 clusters) vs. *Each Item* method using *SLR* for the MovieLens dataset.

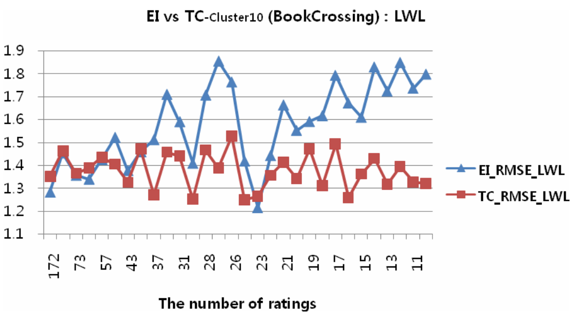


Figure 5. RMSE error rates of *Total Clustering* (with 10 clusters) vs. *Each Item* method using *LWL* for the BookCrossing dataset.

We next studied computational performance of the *TC* method to see how scalable it is to large recommendation problems. To address this issue, we ran all the 9 data mining methods described in Section 2.2 on the MovieLens dataset and clustered all the items in *I*. We ran this problem on a grid at the Stern school for 8 days and would complete only 50% of the job (our job had to be terminated for technical reasons). This example clearly demonstrates that, although superior to *EI*, the *TC* method is very slow and does not scale well to large recommendation problems.

To address the Long Tail Recommendation Problem while achieving reasonable performance results, in Section 3, we present a solution that partitions the itemset *I* into the head and the tail and does clustering *only* in the tail. We also demonstrate that this approach produces smaller error rates than *TC* in some cases.

3. PROPOSED SOLUTION

We next describe how we split the items in *I* into the head *H* and tail *T*, cluster the tail items *T* and build predictive models on the resulting groups. Also, how to split the items into the head and the tail should be decided carefully because it can result in different recommendation error rates. In Section 5.2, we examine good cutting strategies. However, in this section, we assume some arbitrary partitioning of the itemset *I* into the head and the tail.

Having split *I* into head *H* and tail *T*, we cluster items *only* in tail *T* as follows. First, for each item, we compute several derived variables from the transactional variables for that item. For example, for the movie items, examples of these derived variables are the average rating of a movie (*I_aver_rating*), the popularity of the movie (*I_popularity*) and how much customers liked the movie (*I_likablility*). As a result, each item becomes a point in the space of the derived variables. For example, if we have 10 derived variables and 100,000 movies, then each movie becomes a point in the 10-dimensional space that has 100,000 data points in total. Next, we apply standard clustering methods to identify particular clusters in that space (e.g. cluster these 100,000 points in the 10-dimensional space). In this paper we used the *EM* clustering method [7].

We do not use clustering methods in the head *H* and build *individual* predictive models for each item in *H* for the following reason. As Figures 4 and 5 show, the error rates for the *TC* and *EI* methods are relatively close near the origin of these two graphs, which is in contrast to the error rates in the right parts of the graphs. This can be explained by observing that the popular items in the head have already considerable ratings data, unlike the less popular items in the tail. Thus, clustering items in the head should not contribute significantly to the performance of the corresponding data mining models. Therefore, we cluster items only in tail *T* and build individual data mining models for each item in the head. We call this approach the *Clustered Tail (CT)* method.

After we cluster the items in *T*, we build predictive rating estimation models on the resulting clusters. For example, if we clustered movies *A*, *B* and *C* into one cluster, we take the ratings assigned to these three movies, information about the movies and customers and use linear regression to estimate unknown ratings of the three movies.

We applied this process of clustering tail items across a variety of experimental settings and measured by how much it improves performance and solves the *LTRP* problem. We present our experimental settings in Section 4 and the results in Section 5.

4. EXPERIMENTAL SETUP

In this section, we explain the experimental settings used for validating the *Clustered Tail* (*CT*) method, including an overview of the data used, selected variables, data mining methods, performance measurements and statistical tests.

Data. We used two popular datasets in our study MovieLens [5] and BookCrossing [6]. The MovieLens dataset contains 100,000 ratings on the scale of 1 to 5 from 943 customers on 1682 movies. The BookCrossing dataset contains 1,149,780 ratings on the scale of 1 to 10 from 278,858 customers on 271,379 books.

Variables. In order to predict unknown ratings, we used the following derived variables (*DV*) as *independent* variables in our data mining models. Customer-related derived variables *DV* are :

1. *c_aver_rating*: The average rating that customer gives for the items that he or she saw before.
2. *c_quantity*: The number of ratings the customer rated before.
3. *c_seen_popularity*: The average popularity of the items that the customer rated before.
4. *c_seen_rating*: The average rating of the items rated by the customer, each rating being an average of all the ratings provided by all the customers for that item.

5. *c_like_popularity*: The average popularity of the items rated higher than the customer's average ratings.
6. *c_like_rating*: The average rating of the items rated higher than the customer's average ratings, each rating being an average of all the ratings provided by all the customers for that item.
7. *c_dislike_popularity*: The average popularity of the items rated lower than the customer's average ratings.
8. *c_dislike_rating*: The average rating of the items rated lower than the customer's average ratings, each rating being an average of all the ratings provided by all the customers for that item.

Item-related derived variables *DV* are:

9. *I_aver_rating*: The average rating of the item
10. *I_popularity*: The popularity of the item.
11. *I_likablility*: The difference between the rating of the customer and their average rating.

Head/Tail Partitioning. We partitioned the total itemset *I* into the head *(H)* and tail *(T)* parts in several places in our study. In particular, we selected the head/tail cutting points at levels $\alpha = 30$, 50, 70, 90 and 110, where level α means that the items with frequency $> \alpha$ belong to the head *H* and with frequency $< \alpha$ belong to the tail *T* of the item distribution.

Clustering. After partitioning items into head and tail, as described above, we cluster items only in tail *T* as follows. We, first, select variables *I_aver_rating* (9), *I_popularity* (10), and *I_likablility* (11) from the list of derived variables, then we map each item from tail *T* into the 3-dimensional Euclidian space formed from these three variables, and finally apply the *EM* clustering method [7] to the set of the resulting points. The number of clusters in tail *T* used in the *EM* method in our studies are 1, 10, 20, 30, 40 and 50. Note that if the number of clusters is 1, this means that we actually do not do any clustering in the tail.

Data Mining Models. We build the models estimating ratings for the items in each cluster in *T* and also for each item in *H*. In particular, we use Weka [7] and deploy the following data mining models from Weka in our studies that we have already described in Section 2.2: 1) SimpleLinearRegression (*SLR*), 2) Gaussian radial basis function network (*RBF*), 3) Support vector machines (*SVM*), 4) K-nearest neighbours (*KNN*), 5) Locally-weighted learning (*LWL*), 6) Bagging classifier (*Bagging*), 7) DecisionStump tree(*DecisionStump)*, 8) M5P model tree(*M5P*) and 9) 0-R Rule Induction classifier (*ZeroR*).

For each of these models, the dependent variable is Rating and independent variables are selected as follows. We use the following 5 sets of the derived variables (*DV*) (1) through (11) described above[1] as independent variables in these data mining models:

1) Used the whole *DV*
2) Used *DV* – 1, 2, 5, 6, 9, 10, 11
3) Used *DV* – 3, 4, 9, 10, 11
4) Used *DV* – 1, 7, 8, 9, 10, 11
5) Used *DV* – 3, 4, 5, 6, 9, 10, 11

Performance Measurements. We use Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) as performance measures. After building the model, we predict the unknown rating on the holdout sample and calculate the error rates as:

$$(1)\ RMSE = \sqrt{\frac{\sum_{i=1}^{n} e_i^2}{n}} \qquad (2)\ MAE = \frac{\sum_{i=1}^{n} |e_i|}{n}$$

In this section we described the experimental settings. In the next section we present the results of our experiments.

5. EXPERIMENTAL RESULTS

In this section we present the results of the experiments described in Section 4. In Section 5.1, we will focus on comparing the *Clustered Tail* (*CT*) and the *Each Item* (*EI*) methods and demonstrate that *CT* outperforms *EI*. In Section 5.2, we will focus on finding the most appropriate head/tail cutting point and determining the appropriate clustering number for the *CT* method.

5.1 Comparing the Clustered Tail and the Each Item Methods

To compare the *Clustered Tail* (*CT*) and the *Each Item* (*EI*) methods, we performed the paired t-tests across various experimental settings described in Section 4. In particular, we perform these tests across five sets of variables, five cutting points, six clustering groups, nine data mining methods and two measurements. This results in 2700 comparisons of the *CT* and *EI* methods for each dataset.

For each of the 2700 cases, we compute the error rates for the particular predictive model for *each item* in the itemset *I* for the *CT* and the *EI* methods. To check for the statistically significant differences between them, we performed paired t-tests, but only for the items in the tail *T* of the distribution. In other words, if $E = (e_1, \ldots, e_n)$ and $E' = (e'_1, \ldots, e'_n)$ are the error rates for items *I* from the tail *T* for methods *CT* and *EI* respectively, then we perform the paired t-test of sets *E* and E' to detect the statistically significant differences between the two. The reason for considering the errors only in the tail *T*, is that the errors in the head *H* for the two methods *CT* and *EI* are the same, and there is no point in including them in the test.

The t-test comparison results are presented in Table 3. Table 3 shows that for the MovieLens dataset the *CT* model outperforms the *EI* model in 2464 cases at a 95% confidence interval out of the total 2700 comparisons. Similarly, for the BookCrossing dataset, the *CT* model outperforms the *EI* model in 2525 cases.

Table 3. Paired T-test Result (Statistically significant at the 95% confidence interval)

Dataset	$Err._{CT}<Err._{EI}$	$Err._{EI}<Err._{CT}$	$Err._{CT}=Err._{EI}$
MovieLens	2464	116	120
BookCrossing	2526	70	104

Table 3 provides only overall comparison information across the *CT* and the *EI* methods, without showing any specifics for the individual comparisons. Since there are 2700 of them in total, it is impossible to present the specifics on all of them. Therefore, we decided to examine the performance differences between the *CT* and the *EI* models for some selected individual setting (out of the total of 2700 of them).

[1] The numbers to the right specify the variable numbers (1 through 11) as specified in the Variables part of Section 5.

As an example, Figure 6 shows the RMSE error rates for the *CT* and *EI* methods using the whole set of 11 derived variables as the independent variables and building the model using the *DecisionStump* data mining method for the MovieLens dataset. Also, the head/tail cutting point is 110 and the number of clusters is 10 for the *CT* method presented in Figure 6. Similarly, Figure 7 shows the RMSE error rates for the *CT* and the *EI* methods using the whole derived variables as the independent variables and building *Bagging* predictive model for BookCrossing dataset. Also, the head/tail cutting point is 90 and the number of clusters is 10 for the *CT* method presented in Figure 7.

The graphs in Figures 6 and 7 clearly show that the *CT* method outperforms the *EI* method in the tail *T*. These representative examples provide a typical picture of the paired comparisons across the 2700 experimental settings: the *CT* and the *EI* graphs are the same in the head and the initial parts of the tail, but then error rates diverge at the end of the tail, as Figures 6 and 7 show. This divergence accounts for the overwhelming dominance of the *CT* over the *EI* method as is evident from the t-test results reported in Table 3.

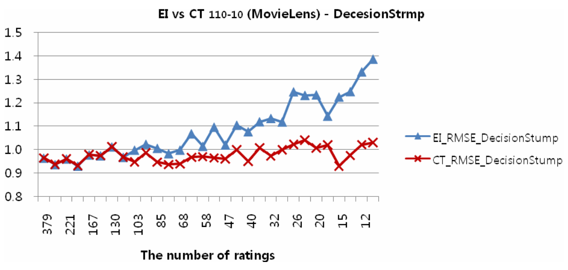


Figure 6. RMSE error rates of *Each Item* and *ClustedTail* using 110-10 methods for the MovieLens dataset.

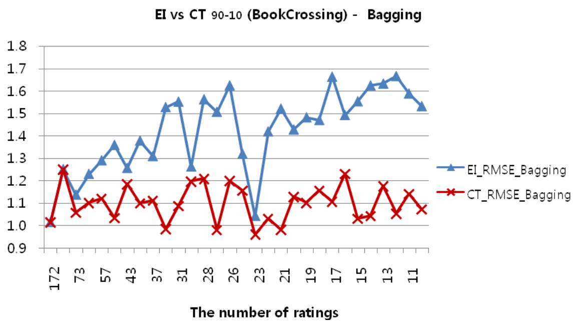


Figure 7. RMSE error rates of *Each Item* and *ClustedTail* 90-10 methods for the BookCrossing dataset.

We next present the *magnitudes* of the performance differences between the two methods. In particular, Figure 8 presents the histogram of the average improvement rate of *CT* vs. the *EI* methods for the RMSE errors for the MovieLens dataset taken across all the 1350 experimental settings described in Section 4, where the Improvement rate is computed as

$$\text{Improvement rate (\%)} = \frac{RMSE_{EI} - RMSE_{CT}}{RMSE_{EI}} \times 100(\%)$$

For example, as Figure 8 demonstrates the *CT* method achieves the 8% improvement rate vs. the *EI* method in terms of the RMSE errors on the MovieLens data in 217 cases. Similarly, Figure 9 presents the histogram of the average improvement rate of *CT* vs. the *EI* methods for the RMSE errors for the BookCrossing dataset.

As Figures 8 and 9 demonstrate, the *CT* method significantly outperforms the *EI* method in most of the cases. Only in 49 cases (out of total of 1350) the differences between the *CT* and *EI* performances are negative for the MovieLens and in 53 cases for the BookCrossing datasets. Also, the performance improvements go as high as 11.78% for the MovieLens and 72.45% for the BookCrossing datasets.

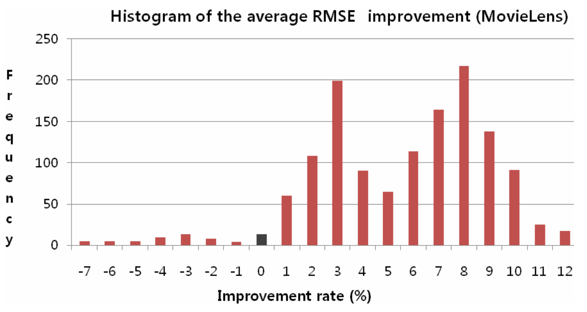


Figure 8. Histogram of the average RMSE improvement rate of *CT* models vs *EI* model for the MovieLens dataset.

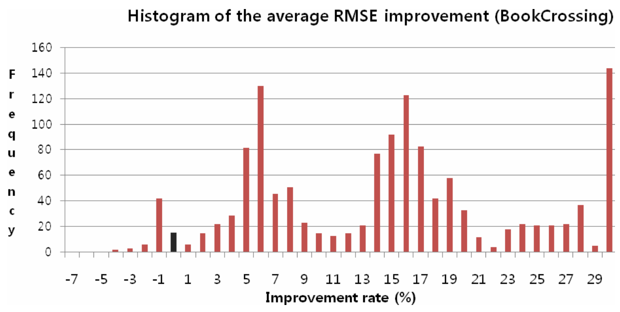


Figure 9. Histogram of the average RMSE improvement rate of *CT* models vs *EI* model for the BookCrossing dataset (the rightmost bar stands for > 29).

All these statistical and visual comparisons of performance results across extensive experimental conditions clearly show that the *CT* method produces significant performance improvements vs. the *EI* method. All this means that clustering each item in the Long Tail *T* indeed produces better recommendations.

In the next subsection, we examine where to cut the itemset *I* into head *H* and tail *T* parts and how to cluster the items in tail *T*.

5.2 Finding the Right Cutting Points

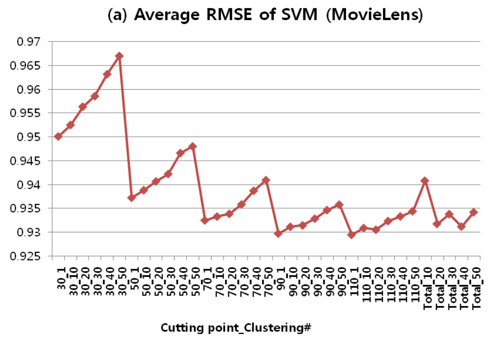
As was observed earlier in the paper, error rates in the tail *T* depend on where we cut the itemset into the head and tail. In this section we empirically study where the best cutting points are and whether it makes sense to partition the items into the head *H* and tail *T* in the first place.

To study this problem, we consider the following five cutting points: *i* = 30, 50, 70, 90 and 110. A cutting point *i* means that the items with the ratings frequency more than *i* belong to the head *H* and the items with frequency less than *i* to the tail *T*. In addition, we also consider the special case of a vacuous head *H,* where all the items appear only in tail *T* (and nothing in the head). Note that this is the *Total Clustering* (*TC*) case described and discussed in Section

2.3. As explained in Section 4, in addition to these cutting points, we also consider different numbers of clusters in tail *T,* ranging from 1 to 50 in the increments of 10 (i.e., 6 different numbers of clusters: $k = 1$, 10, 20, 30, 40 and 50). However, we do not deal with the clustering number $k = 1$ for the Total Cluster (*TC*) case (i.e., placing the whole dataset into one big cluster and having no items in head *H*) because it turns out that this particular case is computationally very expensive. This means that we deal with 35 cases in total in this section (6 cutting points *i* and 6 clustering numbers *k*, minus the special case of $k = 1$ for the Total Cluster case, as just described).

We next compute error rates for each of the 35 cut/cluster combinations described above, and we do it across various experimental settings described in Section 4 (45 such settings for each error rate (RMSE and MAE) and each dataset (MovieLens and BookCrossing) – 180 settings in total). For instance, two examples of such cut/cluster combination graphs are presented in Figure 10 for the MovieLens and two in Figure 11 for the BookCrossing cases.

These four examples demonstrate that error rates are sensitive to the right mixture of the cut/cluster combinations in certain experimental settings. For example, in Figure 10(a), the combination of cutting point 110 and cluster number 1 (110-1) produces the minimum average error rate RMSE = 0.9295 for MovieLens dataset, and in comparison, the worst cut/cluster combination of 30-50 Produces RSME = 0.967, which constitutes 3.75% performance improvement. Similar observations are applicable to the other three graphs presented in Figures 10(b) and 11(a, b).



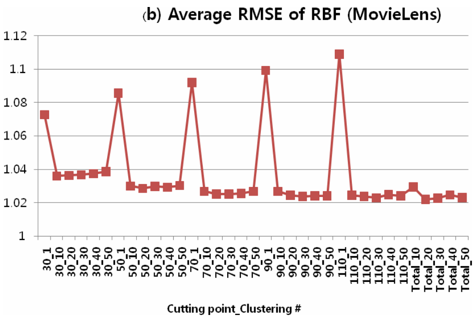
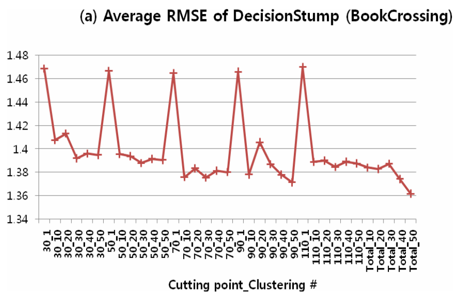


Figure 10. Average RMSE according to the cutting point and the clustering number for MovieLens dataset.

Another important question is to determine for which values of *i* and *k* the error rates reach the minimal levels (such as $i = 110$ and $k = 1$ in Figure 10(a)). We did this analysis across all the 90 experimental conditions for each of the two datasets (while only 4 of them are presented in Figures 10(a, b) and 11(a, b)).

The results of this analysis are presented with the white bars (representing the best model case) in Figure 12 for the MovieLens and in Figure 13 for the BookCrossing datasets. Both Figures 12 and 13 show that in a significant number of cases, the *TC* solution constitutes the best case scenario producing the minimal error rates. For example, Figure 11(a) clearly demonstrates this point since the minimal error rate is achieved for the Total_50 case.

However, if we examine Figure 10(b), we can see that the difference between the smallest error rate achieved for Total_20 and the second best point of 110-30 is highly insignificant (1.0218 vs. 1.0227 in this case). This means that, even though the *Total Clustering (TC)* is theoretically the best solution, in practice it may not be the case since it is usually computationally very expensive, while it achieves highly insignificant improvements in error rates. Therefore, it may be better to replace such *TC* model with the second best, but much cheaper *CT* model. We next compare performances of the Best vs. Second-best models by applying the paired t-tests to the overall performances of the corresponding models. If the differences are statistically insignificant at the 95% confidence level, we replace the best-performing *TC* with the second-best-performing *CT* model. Then the Practical best performing model (*Practical solution)* is: select the best-performing model *TC* if the second-best one is significantly worse. Alternatively, if the performance differences are statistically insignificant, then select the second best *CT* model. The histograms of the Practical solutions are shown with black bars in Figures 12 and 13 for MovieLens and BookCrossing datasets respectively.



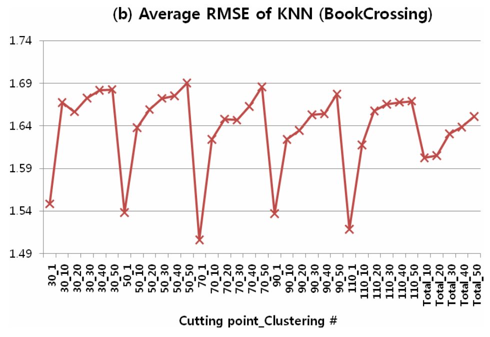


Figure 11. Average RMSE according to the cutting point and the clustering number for BookCrossing dataset.

As Figures 12 and 13 demonstrate, there are significantly fewer best-performing Practical models for the *Total Clustering* (*TC*) case. In fact, most of the practically best-performing models fall within the middle region in both figures (Figures 12 and 13).

This result demonstrates that *partitioning the itemset I into head and tail and clustering items in the tail is often the best solution* dominating the *Total Clustering* in practice.

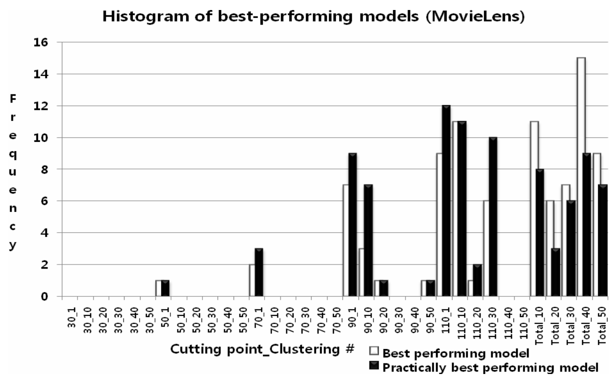


Figure 12. Histogram of the best model for MovieLens dataset.

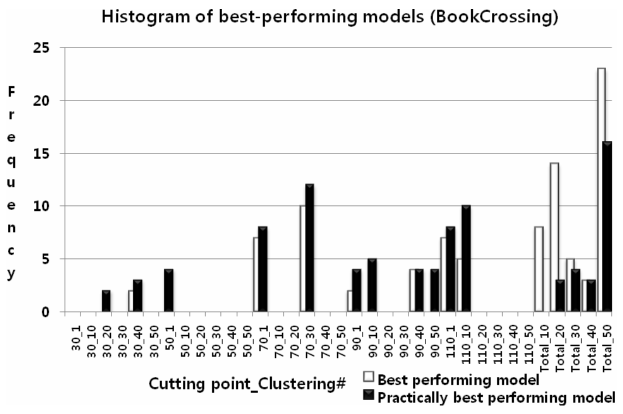


Figure 13. Histogram of the best model for BookCrossing dataset.

6. CONCLUSION

In this paper, we identified the Long Tail Recommendation Problem (*LTRP*) responsible for the increased error rates of the items in the Long Tail of the item distribution. We showed that a simple item grouping method, although effective at reducing these rates, is not practical since it does not scale well to large problems.

Therefore, we proposed to partition itemset *I* into the head *H* and the tail *T* parts and cluster the items in the tail *T*, while keeping the basic *Each Item* (*EI*) approach for the head items *H.* We showed that this solves the *LTRP* problem in the sense that the error rates of the items in tail *T* are significantly lower than for the basic *EI* method. Moreover, we also showed that this approach of the head/tail partitioning of itemset *I* and grouping items in *T* is more scalable than grouping the whole itemset *I.*

We also studied the problem of selecting good head/tail cutting points α and identifying the right numbers of item clusters in the tail. We showed that in some cases grouping of the whole itemset *I* (no head/tail partitioning) results in the best performance results. However, in many cases in our study, it turns out that the best cutting point α lies somewhere in the *middle* of the itemset distribution histogram and therefore the *TC* method *does not* always constitute the best approach. This also means that a good cutting point α and the right number of tail clusters need to be selected carefully since these parameters affect the performance of recommender systems significantly and the good choices of their values depends on various parameters that vary across different datasets and recommendation approaches.

In summary, the contributions of this work lie in showing that a) the item-based Long Tail of the ratings distribution does matter; b) the items in the Long Tail can be used productively by clustering them into various groups; c) the practically best head/tail cutting points often lie in the middle of the range, as Figure 13 shows, and empirically finding such cutting points.

In the future, we plan to address scalability issue since some of our experiments took a long time to run, especially when the head/tail cutting point α was skewed more towards the head and the number of clusters in the tail was small. We would also like to develop incremental algorithm for determining optimal splitting points when new rating and other data about items and users is added or changed dynamically. We would also like to combine our *CT* method with other recommendation approaches, such as collaborative filtering, and see if this combined method improves performance even further. Finally, we studied the binary splitting problem of the itemset into the Head and Tail. In the future, we would like to consider multiple (non binary) partitioning of the item base, each partition having its own grouping methods.

7. REFERENCES

[1] Schein, A., Popescul, A., Ungar, L. and Pennock, D. 2002. Methods and Metrics for Cold-Start Recommendations. Proc. of the 25th ACM SIGIR Conference.

[2] Anderson, C. 2006. The Long Tail. Hyperion press.

[3] Fleder, D.M., and Hosanagar, K. 2008. Blockbuster Cultures Next Rise or Fall: The Impact of Recommender Systems on Sales Diversity. NET Institute Working Paper No. #07-10.

[4] Hervas-Drane, A. 2007. Word of Mouth and Recommender Systems: A Theory of the Long Tail. NET Institute Working Paper No.07-41, November 2007.

[5] http://movielens.umn.edu.

[6] http://www.bookcrossing.com.

[7] Witten, I.H., and Frank, E. 2005. Data Mining: Practical machine learning tools and techniques with Java implementations. Morgan Kaufmann.

[8] Truong, K.Q., Ishikawa, F., Honiden, S. 2007. Improving Accuracy of Recommender System by Item Clustering, IEICE TRANSACTIONS on Information and Systems, E90-D-I(9).

[9] Ungar, L.H. and Foster, D.P. 1998. Clustering Methods for Collaborative Filtering. Proceedings of the Workshop on Recommendation Systems. AAAI Press.