Clustering Approaches for Top-N Recommender Systems

Nicolás Torres
Universidad Técnica Federico Santa María
Av. Vicuña Mackenna 3939, San Joaquín
Santiago, Chile
nicolas.torresr@usm.cl

Marcelo Mendoza
Universidad Técnica Federico Santa María
Av. Vicuña Mackenna 3939, San Joaquín
Santiago, Chile
marcelo.mendoza@usm.cl

ABSTRACT

Collaborative filtering (CF) is one of the most successful techniques used in recommender systems. It is based on the idea that people often get the best recommendations from someone with similar tastes to themselves. Broadly, there are memory-based and model-based CF techniques. Neighborhood-based CF uses similarity measures to compute the proximity between users, a kind of methods known as User-based CF (UBCF). UBCF, while effective, suffers from scalability problems as the database grows. To address the scalability issue, clustering-based CF algorithms constraint the seek of users within small user clusters instead of the entire database. Then clusters can naturally be distributed into different partitions scaling up in terms of the number of users the system can handle. However there is a trade off between scalability and prediction performance. Whilst the number of users and items included in a cluster solution increases, the performance in terms of precision of a clustering-based CF algorithm decreases. We present a novel approach that combines the advantages of UBCF and cluster-based CF methods by introducing a cluster-based distance function used for neighborhood computation. In our approach, clusters generated from the training data provide the basis for neighborhood selection. Then, to expand the search of relevant users/items we use a novel measure that is able to exploit the global cluster structure to infer user's distances. In addition, our measure allows to handle the incorporation of near neighbors outside cluster frontiers. Empirical studies on widely known benchmark datasets suggest that our proposal is feasible.

CCS CONCEPTS

•**Information systems** → Retrieval tasks and goals;

KEYWORDS

Recommender systems, clustering approaches

ACM Reference format:
Nicolás Torres and Marcelo Mendoza. 2017. Clustering Approaches for Top-N Recommender Systems. In *Proceedings of Large Scale Recommender Systems jointly with ACM RecSys, Como, Italy, August 2017 (LSRS'17),* 8 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Clustering techniques work by identifying groups of users who appear to have similar preferences. Clustering over vector representations of users is the most common approach used for this purpose. The representation of a user as a vector of preferred items is a widely used approach. Then, prototype-based clustering algorithms (e.g. $\mathcal{K}$-means) can be applied over user vectors to create clusters of users. Once the clusters are created, predictions for unseen items for a target user can be made by averaging the ratings of the other users in that cluster. This approach usually produce less-personalized recommendations than other methods and in most cases cluster-based recommendations have worse accuracy than near neighbors algorithms.

In this article we study the use of different prototype-based clustering algorithms in recommender systems ($\mathcal{K}$-means, $\mathcal{K}$-medians and spectral clustering). Then, instead of constraint the search of neighbors to cluster frontiers, we define a novel distance function able to search for similar users outside the frontiers of a given cluster at the cost of introducing an additional operation into the neighborhood computation. We will show that UBCF allows good results in terms of precision and recall over small datasets as MovieLens and BookCrossing. However, we will show that cluster-based CF outperforms UBCF when the data are sparse (last.fm).

UBCF based on k-Nearest Neighbors (k-NN) has demonstrated to be very effective, but is reputable for being feeble in two major areas: (i) It is computationally expensive for large-scale datasets; (ii) Its performance according to a given distance function (e.g. normalized Euclidean distance) decreases when dimensionality increases. The use of clustering allows to reduce the impact of data sparseness a common characteristic of high dimensional datasets, as cluster prototypes can be used as features for user-user distance computation. In addition, data summarization techniques can help recommender systems to limit data redundancy. As UBCF based on k-NN minimizes the intra distance in each neighborhood, it over exploits data locality. The inclusion of items retrieved from near clusters instead of near users can help to handle the exploitation/exploration balance.

The main contribution of our article is to study the impact of cluster models on Top-*N* recommender systems. In specific, two contributions arise from our proposal:

(1) We use cluster prototypes to calculate user neighborhoods combining intra and inter neighbors, limiting the impact of sparseness on distance functions.
(2) We define a new distance function able to handle neighborhood computation inside and outside cluster frontiers.

Clustering-based CF methods have better scalability than typical CF methods because they make recommendations within much smaller clusters rather than the entire customer base. As users are grouped into clusters, it is possible to distribute the clusters along different data partitions. Note that the construction of the clustering solution runs off line. However, its recommendation quality

LSRS'17, Como, Italy
2017. 978-x-xxxx-xxxx-x/YY/MM...$15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

is generally low. We argue that it is possible to improve the quality of the recommendations by including similar neighbors from outside the cluster.

Organization. The rest of the paper is organized as follows. In Section 2 we present a review of the literature. In section 3 we introduce our proposal. In Section 4 we present our validation setup. In Section 5 we evaluate our proposal and experimental results are discussed. We conclude in Section 6 highlighting our study's main achievements.

2 RELATED WORK

Collaborative filtering (CF) based methods for recommender systems may be classified into memory-based and model-based strategies [1]. Memory-based CF methods use proximity functions between users to estimate ratings on unseen items. For a given user and item pair, the algorithm retrieve the group of users who previously reviewed the item, using their ratings to elaborate the recommendation for the user. The key idea is to weight each rating by the proximity of each user to the target user. This idea was firstly applied by GroupLens [10], exploring the use of the Pearson correlation measure to estimate the rating of a given item-user pair. Analogously, recommendations can be conducted over groups of similar items previously seen by the target user, estimating rates for new items that are close to the old ones [12]. The first kind of methods is known as User-Based CF (userkNN) and the second one is known as Item-Based CF [5]. Both UBCF and IBCF methods over exploits data locality, because user or item neighborhoods tend to include very similar neighbors, introducing data redundancy in the rating estimation step. In addition, these methods need to include data structures (e.g. R-trees or kd-trees) to handle neighborhood computation during execution, forcing the allocation of memory areas of large size.

Data can be partitioned using clustering algorithms to distribute clusters of users across different data partitions scaling up to large scale datasets. This kind of algorithms use a clustering algorithm for cluster computation, constraining the retrieval of each neighborhood the each cluster. The first CF algorithm using this idea applied a clustering algorithm on the user-item ratings database to split the dataset in clusters of users [11]. The clustering algorithm may generate fixed sized partitions, or based on some similarity threshold it may generate a requested number of partitions of varying size. Then, a UBCF algorithm is applied to produce recommendations, constraining the calculation of each user neighborhood to the frontier of each cluster. Experimental results show that this proposal is feasible but at the cost of retrogress the precision of the recommendations. We will refer to this algorithm as PureCL. A slightly different idea was explored by Xue et al. [15] where clustering was used to boost rate estimation. In this work, the distance between the target user and its nearest neighbors was modified replacing missing vector entries by rate averages in each neighbor's cluster. Recently, a new similarity score for neighborhood computing based on clustering was proposed [14]. The similarity score named LiRa uses clustering to compute the likelihood ratio between rate differences of users that belongs to the same clusters and differences produced by chance. Experimental results show that the similarity function behaves well in sparse datasets.

Kelleher et al. [9] introduced a CF method based on hierarchical agglomerative clustering algorithm (HAC) to obtain data partitions. The method comprises summary rating information derived from the HAC of the users. They show that the proposal is efficient in terms of computational time and acceptable in accuracy. George et al. [6] use co-clustering to conduct data partitions in a simultaneous clustering approach over users and items. Experimental results show that the proposal is feasible improving accuracy in some cases. Iterative clustering between items and users was explored by Jiang et al. [8] reinforcing the clustering process. Experimental results show that the conduction of two simultaneous clustering strategies (item and user-based) may alleviate the data sparsity problem in recommender systems. The location of each user was exploited by Das et al. [3] conducting spatial clustering to create user partitions. A weighted Voronoi-based approach was used to achieve this goal, constraining neighborhood computing to the frontiers of each spatial cluster. Genre-based user clustering on movie's databases was addressed using DBSCAN [4]. A maximum of four genres per user was allowed to conduct genre-based clustering, creating user partitions according to their tastes. Then, neighborhood computing was constrained to each cluster, showing that genre-based clusters are useful for improving the quality of the recommendations.

In summary, clustering can be considered as a pre-processing stage used to create data partitions, exploiting each partition to constrain the computation of each neighborhood or to handle missing rating entries by using the cluster as a rate smoother. In a second step, this kind of algorithms uses standard methods to produce the lists of items, being CF based methods the most common techniques used for this purpose. The state of the art lacks of proposals based on the combination of intra an inter clusters data objects for neighborhood computation, which is the main purpose of our study.

3 THE PROPOSED APPROACH

3.1 Definitions and Notations

Now we introduce some definitions to construct our proposal. Let $U = \{u_1, \ldots u_N\}$ be a set of users and $V = \{v_1, \ldots, v_M\}$ a set of items. A rating matrix R created from $U \times V$ with N rows and M columns encompasses a collection of pairwise user-item rates, denoting by $R_{i,j}$ the corresponding rate for u_i and v_j. If user-item rating pairs are inferred from implicit feedback sources (for instance, seen/unseen user-item pairs), each $R_{i,j}$ may be 1 for seen items, 0 otherwise. If item ratings are retrieved from explicit feedback sources (e.g., graded human judgments), $R_{i,j}$ may be defined as an ordinal variable. Note that only a subset of all the entries of R is observed, and in general, a great proportion of entries in R in unobserved. The task of a recommender systems is to predict the rating of each unobserved entry of R to produce recommendations to each user.

Let $U(v)$ be the group of users who has seen/preferred an item v. Let u be the target user and v an unseen item by u. A user-based CF method (UBCF) estimates $R(u, v)$ from R as follows:

$$\hat{R}(u,v) = \sum_{u' \in U(v)} \mathrm{Sim}(u, u') \cdot R(u', v),$$

where $\text{Sim}(u, u')$ is a pairwise similarity function for the u-u' pair.

In general, UBCF methods constraint $U(v)$ to the nearest neighbors of u, avoiding the inclusion of distant users to u, improving the quality of the estimation $\hat{R}(u, v)$. In addition, there are extensions to this expression that includes user bias correction, as mean-centering or Z-score. A detailed review of these variants is provided in [13].

3.2 The Proposed Distance

Now we introduce our user clustering-based method (userCL) for Top-N recommendations, based on a novel distance function that can retrieve near-neighbors outside the frontiers of a cluster. Our distance function works over cluster prototypes, assuming that for each cluster the system has a prototype that represents the cluster (e.g. a centroid).

Suppose that we have run a hard clustering algorithm between users over the item vector space achieving $\mathcal{K}$ disjoint clusters. Let C_x be the prototype of the cluster to which the object x belongs. The central idea is to modify the distance function between a pair of objects (users) by considering how far are the objects to their prototypes, allowing to recover near neighbors outside the frontiers of a cluster if the objects are far from the prototype.

If two objects belong to the same cluster, our distance function is defined as the distance between the objects. However, if both objects belong to different clusters, their distance is calculated by computing the distance between those centroids. The idea is to force a separation between objects in a high dimensional feature space, avoiding the direct calculation of the distance between objects that belong to different clusters. If the clustering algorithm is able to produce good clusters, the distance between cluster prototypes will be high. Then, by measuring distances between prototypes, points that belongs to different clusters will be more separated, even if both cluster frontiers are close each other. Accordingly, we define our distance function $D(x, y)$ for a given x-y pair of objects by:

$$D(x,y) = \begin{cases} d(x,y) & \text{if } C_x = C_y, \\ d(C_x, C_y)^{exp} & \text{if } C_x \neq C_y, \end{cases} \tag{1}$$

where $d(x, y)$ is a distance function between the x-y pair of data objects (e.g. the normalized Euclidean distance). For simplicity we will assume that the basic distance function $d(x, y)$ ranges in $[0, 1]$, and accordingly corresponds to a normalized distance as the Ochini distance (the complement of the Cosine similarity) or the normalized Euclidean distance.

Note that the distance between centroids $d(C_x, C_y)$ considers an exponent exp. We defined three variants for exp:

[1] Multiplicative variant:

$$exp = 1 - d(x, C_x) \cdot d(y, C_y) \tag{2}$$

[2] Additive variant:

$$exp = 1 - \frac{1}{2}(d(x, C_x) + d(y, C_y)) \tag{3}$$

[3] Additive-sigmoid variant:

$$exp = 1 - \left(1 + \frac{1}{\alpha} e^{-\beta(\frac{1}{2}(d(x,C_x)+d(y,C_y))-1)}\right) \tag{4}$$

where α and β are strictly positive real parameters.

The rationale behind each variant of exp is explained below.

[1] Multiplicative variant ($C_x \neq C_y$)

If x is close to C_x OR y is close to C_y
$d(x, C_x) \cdot d(y, C_y) \to 0 \Rightarrow D(x, y) \approx d(C_x, C_y)$
If x is far from C_x AND y is far from C_y
$d(x, C_x) \cdot d(y, C_y) \to 1 \Rightarrow D(x, y) \to 1$

The idea behind the multiplicative variant is to penalize the distance between objects in distinct clusters if both objects are far from their respective prototypes. If one or both objects are close to their respective prototypes, the distance between the objects is approximated by the distance between the centroids. On the other hand, if both objects are far from their respective prototypes, the distance will be close to 1.

[2] Additive variant ($C_x \neq C_y$)

If x is close to C_x AND y is close to C_y
$\frac{1}{2}(d(x, C_x) + d(y, C_y)) \to 0 \Rightarrow D(x, y) \approx d(C_x, C_y)$
If x is far from C_x AND y is far from C_y
$\frac{1}{2}(d(x, C_x) + d(y, C_y)) \to 1 \Rightarrow D(x, y) \to 1$

The idea behind the additive variant is that if both objects are close to their respective prototypes, the distance between the objects is approximated by the distance between the prototypes. On the other hand, if both objects are far from their respective prototypes, the distance will tend to 1. Thus, the additive variant is a more demanding version of the multiplicative variant, because it approximates the distance between the objects by the distance of their prototypes iff both objects are close to their respective prototypes.

[3] Additive-sigmoid variant ($C_x \neq C_y$)

If x is close to C_x AND y is close to C_y
$\text{Sigmoid}(\frac{1}{2}(d(x, C_x) + d(y, C_y))) \to 0 \Rightarrow D(x, y) \approx d(C_x, C_y)$
If x is far from C_x OR y is far from C_y
$\text{Sigmoid}(\frac{1}{2}(d(x, C_x) + d(y, C_y))) \to 1 \Rightarrow D(x, y) \to 1$

The last variant of exp is an extension of the additive variant that includes a sigmoid function, helping to increase the penalization in the border of the original pairwise distance range. The argument of the sigmoid function is the additive factor, stretching the extremes of the additive factor.

3.3 The UserCL method

UserCL starts by clustering each user in a cluster of users according to some prototype-based clustering algorithm with $\mathcal{K}$ prototypes. Then, based on Equation 1, we compute the distances for a given target user to their neighbors, selecting the k-nearest neighbors (kNN) according to one of the three variants proposed. Our distance function can be efficiently implemented in a distributed system, starting neighborhood computing inside the cluster of the target user. Then, if the number of retrieved neighbors from the cluster is less than k, we may require to the closest cluster/partition the retrieval of more neighbors, and so on. Note that this strategy does not guarantee the retrieval of the closest k neighbors of the target user. On the other hand, a less efficient but exact neighborhood distributed computing solution can be implemented. As our distance function avoids the computation of the basic distance function $d(x, y)$ between every pair of users replacing these comparisons by the distances of each point to its prototype, it reduces data sharing between partitions for neighborhood computing. Note that prototypes work as distance proxies for users in different clusters/partitions. Then, to compute the distance between users in different clusters/partitions, we just need to locally compute the distance of each object to its prototype, sharing the value by message passing.

Cluster size is a key element of our approach. Our idea is to constraint cluster sizes (in average) to force the retrieval of some neighbors from outside the cluster frontiers. To do this, we will need to tune the number of clusters $\mathcal{K}$ and the neighborhood size k to force that in average $k \geq \frac{N}{K}$, where N is the number of users recorded in the dataset. The ability to produce clusters balanced by sizes will depend on the type of clustering algorithm used for this purpose.

UserCL works in a tandem with UBCF, using the ratings of the users in $U(v)$ constrained to the neighborhood of u to estimate $R(u, v)$. Note that our distance function can be applied to IBCF methods. In this article we focus the analysis of UserCL only to UBCF methods, postponing the exploration of our proposal on IBCF methods for a future study.

To perform user clustering, we start using k-means. Later, we estimate a low rank approximation of R to create a compact representation of each user performing spectral clustering over a low dimensional space. Both algorithms were tested using the normalized Euclidean distance and Cosine similarity (Ochini distance) to compare the effect of the distance function over each dataset. Finally, we also explored the impact of k-medians on our proposal, favoring the robustness of the algorithm in front of the presence of outliers.

In this paper, we implemented our proposal in a multi-core computer allocating data partitions into main memory. We compared our algorithm to state of the art CF algorithms in the same system, allocating datasets into main memory to conduct a fair comparison. We ran our experiments in a HP400G2 5PD SFF with 16GB of memory and 500GB of hard disk.

4 EXPERIMENTAL SETUP

4.1 Datasets

To evaluate userCL, we used four different datasets that have been widely used in recommender systems evaluation. The first and second datasets (ML100K and ML1M) corresponds to movie ratings and were obtained from the MovieLens[1] research projects. The third dataset was BX, a subset from the Book-Crossing[2] dataset, in which each user has rated at least 50 items and each item has been rated by at least 30 users and at most 300 users. Finally, we use a Last.fm dataset that corresponds to a data sample extracted from the Last.fm online music system[3]. The dataset recovered from Last.fm in 2009 comprises a set of nearly 1,000 users. The dataset comprehends a long list of preferences over a large collection of artists (more than 100K artists were included in the original dataset). We selected users constraining the sampling process to the set of users who listened at least 50 different artists. Then, we achieved a dataset of 495 users with at least 50 different artists listened over a collection of almost 40K artists. This dataset uses the number of reproductions per artists as a source of implicit feedback, then when a given user reproduces more songs for a given artist, the dataset records a high rating value for the user-artist pair. The importance of the Last.fm dataset is the long tail effect of preferences over artists. The dataset exhibits the effect of the Zipf law over user preferences, with a very few artists concentrating the majority of the reproductions, and a long tail of artists with only a very few reproductions, producing a low density dataset. A detailed analysis of the long-tail effect on recommendations was discussed by Celma [2].

A summary of data statistics for the datasets used in this article is shown in Table 1.

Table 1: Statistics of the data sets

	ML100K	ML1M	BX	Last.fm
Users (N)	943	6040	2963	495
Items (M)	1664	3706	7064	39,814
Ratings ($nnzs$ of $\mathcal{R}$)	99,392	1,000,208	244,403	95,711
min $\lvert\mathcal{R}_u\rvert$[a]	20	20	20	50
Density[b]	6.33%	4.47%	1.17%	0.48%

[a] Minimum number of ratings per user.
[b] Non-zero entries ($nnzs$ of R) over $M \times N$.

As Table 1 shows, the datasets used in our experiments considers users with at least 20 rated items (for Last.fm this number grows to 50). Accordingly, cold start scenarios were not evaluated in our setting.

For relevance evaluation in Movie Lens we fixed a relevance threshold at 4, with rates ranging in 1 to 5. The relevance threshold for BookCrossing was fixed at 8, with rates ranging in 1 to 10. Finally, as last.fm includes implicit feedback (listened/unlistened), we used a binary relevance threshold in this dataset.

4.2 Evaluation Methodology and Metrics

We applied 5-fold cross validation to evaluate the performance of recommendation methods. In each run, each of the datasets was split using the 80% of the data for training and the remaining 20% for testing. From each user in the testing set, 15 rated items were randomly selected and provided to the recommender system as user history, testing the algorithm in the rest of the items. Then,

[1]http://grouplens.org/datasets/movielens/

[2]http://www2.informatik.uni-freiburg.de/~cziegler/BX/

[3]https://www.last.fm/

for each user in the testing set a top-N ranked list of recommended items was generated by each method. Results reported in Section 5 shows evaluations for top-N lists with $N = 1$ and $N = 10$. As recommendation measures we used Precision, Recall and nDCG, a set of measures widely used in recommender systems evaluation.

All the algorithms compared in Section 5 were implemented in R recommender lab [7].

5 EXPERIMENTAL RESULTS

In this section, we present the evaluation of userCL comparing its performance with user k-Nearest Neighbors Collaborative Filtering (userkNN) and with the Clustering-Based Collaborative Filtering method (PureCL) discussed in Section 2. We split the results into three sets of experiments. In the first set of experiments we compared the methods using the normalized Euclidean distance and the Cosine similarity on ML100K. We compared the runtime of userCL over three different clustering algorithms on ML100K: k-means, spectral clustering and k-medians. Based on the results of these experiments we selected the best clustering algorithms to conduct a more extensive evaluation over the remaining datasets. Then, in a second set of experiments, we explored distance distribution for ML100K and Last.fm, two datasets that differ in density, discussing the impact of distance histograms on kNN computation and showing how our distance function works over the different datasets used in these experiments. Finally, we conducted an exhaustive comparison among all methods over the four datasets in terms of Precision, Recall and nDCG for top-1 and top-10 lists, showing model and testing times to discuss the efficacy/efficiency trade-off.

We used $k = 50$ and $\mathcal{K} = 20$ in our experiments. These values allow to evaluate our proposal under different data balance conditions. A balanced solution in terms of cluster size will produce clusters with 47, 302, 148 and 25 members in each cluster in ML100K, ML1M, BX and Last.fm, respectively. To limit the effect of cluster size in our results, we ran distance computing for every pair of users in each dataset, using brute force. Due to lack of space, we omit in this paper results obtained with different values of k and $\mathcal{K}$.

5.1 Preliminary efficiency results on ML100K

We compared userkNN and userCL using different clustering algorithms to measure the impact of the clustering technique on computational costs. We measured the performance in terms of real time elapsed for each algorithm splitting the results between training time and testing time for top-10 recommendations. These results are shown in Table 2.

Table 2: Top-10 efficiency on ML100K using three different prototype-based clustering algorithms

method	ML100K						
	clust	d	mt	tt	d	mt	tt
userkNN	–	Euc	0(s)	21.26(m)	Cos	0(s)	1.76(m)
userCL	kmns	Euc	1.14(s)	57.16(m)	Cos	0.92(s)	20.57(m)
userCL	spect	Euc	5.33(s)	44.06(s)	Cos	5.25(s)	12.1(s)
userCL	kmed	Euc	6(s)	42.4(s)	Cos	6.02(s)	9.04(s)

Table 2 shows the results of this experiment. Columns corresponding to mt (model time) and tt (testing time) present the time used by model learning (clustering algorithm with training users) and recommendation (Top-N recommendations for testing users), respectively. Three clustering algorithms were evaluated in this experiment, k-means, spectral clustering and k-medians, whose results are showed in the last rows of the table. Spectral clustering and k-medians testing achieve the best running times for this experiment. Note that in the case of Cosine similarity, the speed up is 10x (testing time on userkNN versus userCL on spectral clustering). Curiously, testing times for userCL on k-means are worse than the ones obtained by userkNN. In ML100K, the clustering solution provided by k-means adds an overhead that makes our solution more expensive than userKNN in terms of computational costs, due to the high imbalance in cluster size produced by the presence of outliers. Accordingly, we discarded k-means for userCL in the rest of this section. On the other hand, k-medians is more robust to the presence of outliers achieving a better balance in terms of cluster size, and then, its computational cost is less than the one obtained using k-means. Thus, spectral clustering and k-medians will be selected as clustering algorithms for userCL.

5.2 The effect of the distance function on KNN computation

Now we explore the effect of the distance function on KNN computation for the two most different datasets in terms of density. In Figure 1 we show the distributions of distance and similarity functions on ML100K and Last.fm. To compute these histograms, we calculated the distances between every pairs of users in each dataset. Each bin in each histogram indicates the number of pairs that reaches the distance value consigned by the bin.

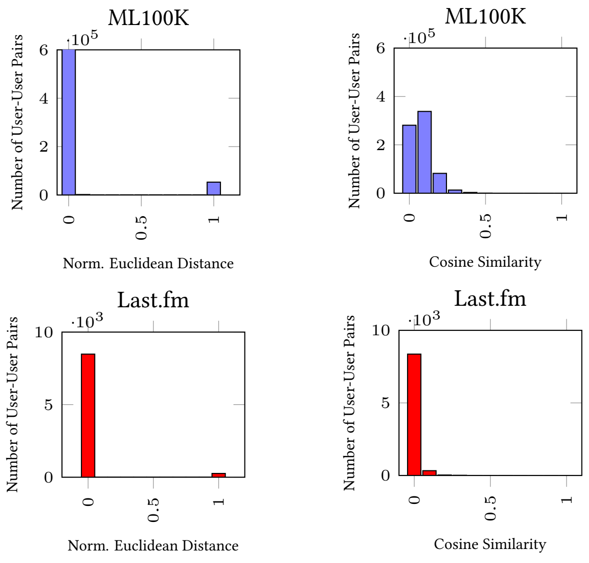

Figure 1: Histograms of proximity between User-User Pairs.

Figure 1 shows that the normalized Euclidean distance gets a bad distribution of pairwise distances for both datasets, with only a small fraction of pairs at distance 1 and a great majority of pairs at zero distance. On the other hand, when the Cosine similarity function is used, the distribution of pairwise points on each histogram improves, being this effect more notorious on ML100K. In Last.fm, the improvement achieved using Cosine similarity is less noticeable, illustrating that the effect of the density in this dataset is only marginally improved by replacing the normalized Euclidean distance with the Cosine similarity.

In Figure 2 we show the pairwise distance histograms achieved by our distance function. Two variants were used in this evaluation, using as a basic distance between points the normalized Euclidean distance and the Cosine similarity function. Results for the three variants proposed are shown in the figure, on both datasets (ML100K and Last.fm).

Figure 2 shows the histograms for ML100K and Last.fm in purple and red colors, respectively. According to the histograms showed on ML100K, the use of the Cosine similarity produces better distributions than the ones obtained using the normalized Euclidean distance for the three variants proposed. Our function over the normalized Euclidean distance produces bins around 0.5, with only two or three bins concentrating the great majority of pairs. When our distance is used over the Cosine similarity, the distribution of pairs around 0.5 improves, with 7 or 8 bins concentrating the great majority of pairs. In ML100K, the best distribution is achieved using the multiplicative variant of our function. Additive and Sigmoid variants bring closer the pairs, with more bins close to the zero distance. When the distribution of pairs was calculated on Last.fm (histograms in red) the effect of the use of Cosine similarity was more noticeable. In these cases, the three histograms have more than 8 bins, outperforming the histograms achieved by the normalized Euclidean distance.

5.3 Top-N performance over the four datasets

Now we report the results over the four datasets used for the evaluation. Table 3 shows these results for PureCL and userCL, methods that were tested using spectral clustering and k-medians. Results based on normalized Euclidean distance and Cosine similarity were reported for the four datasets in terms of Precision, Recall and nDCG. Top-N lists were produced for Top-1 and Top-10 recommendations. Bold fonts indicate the best results for each dataset.

Table 3 shows that the best results for ML100K were achieved by userkNN using Cosine Similarity. In this dataset, the difference between the results obtained using the normalized Euclidean distance and Cosine similarity is very small and the greater difference in terms of Precision and Recall was achieved using userKNN on Cosine similarity, where more than 20% precision and recall points were recorded on Precision Top-1 and Top-10. No significant differences were achieved on nDCG.

Table 3 shows that the best results on ML1M were also achieved by userkNN using Cosine similarity. The main difference was achieved in Precision and Recall but all the methods achieved very similar results using nDCG as evaluation measure. In this dataset, no significant differences were registered between normalized Euclidean distance and Cosine similarity on userCL. In addition, the

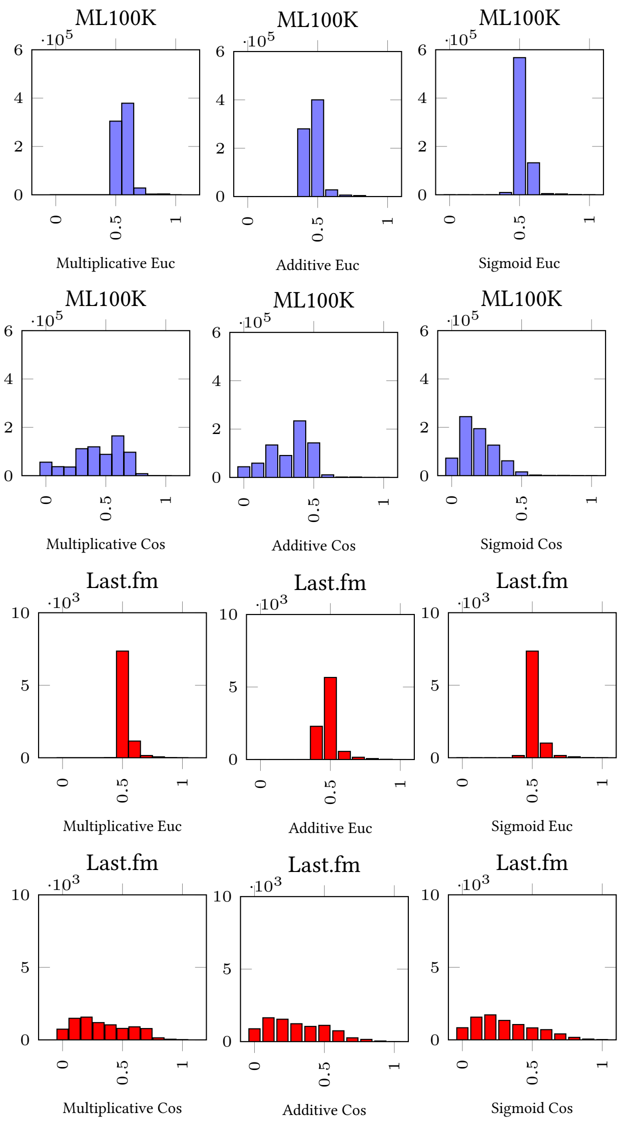

Figure 2: Histograms of proximity between User-User Pairs in userCL

three variants of userCL achieved very similar results in terms of Precision, Recall and nDCG.

Table 3 shows that the results on BX are very poor for all the methods evaluated. Once again, the best result was achieved by userkNN using Cosine similarity. However, when the methods are compared using nDCG, the table shows a slight difference in favor of userKNN, with only 1% or 2% points of difference between userCL and userkNN. In this dataset, the poorest result was achieved by PureCL.

The three first datasets analyzed (ML100K, ML1M and BX) show the same tendency. The poorest results were achieved by PureCL, confirming the presence of a gap in terms of Precision and Recall with userkNN. When userCL was compared, the gap with userkNN

Table 3: Top-N Rec. Performance (@1, @10) on all the datasets

method			ML100K								ML1M							
	params		Precision		Recall		nDCG				Precision		Recall		nDCG			
	clust	*d*	@1	@10	@1	@10	@1	@10	mt	tt	@1	@10	@1	@10	@1	@10	mt	tt
userkNN	–	Euc	0.38	0.26	0.01	0.07	0.82	0.82	–	21m	0.40	0.31	8e-3	0.06	0.86	0.85	–	29h
PureCL	spect	–	0.33	0.25	0.01	0.07	0.80	0.81	4.1s	8.9s	0.36	0.29	8e-3	0.05	0.84	0.84	4m	4.6m
userCL[1]	spect	Euc	0.51	0.30	0.03	0.14	0.81	0.80	4.3s	40s	0.54	0.39	0.02	0.10	0.86	0.84	4m	21m
userCL[2]	spect	Euc	0.50	0.32	0.03	0.14	0.82	0.81	4.3s	40s	0.56	0.41	0.02	0.10	0.86	0.85	4m	21m
userCL[3]	spect	Euc	0.54	0.31	0.03	0.14	0.82	0.80	4.3s	40s	0.54	0.40	0.02	0.10	0.86	0.85	4m	21m
PureCL	kmed	–	0.36	0.27	0.02	0.09	0.80	0.81	4s	8.4s	0.36	0.28	7e-3	0.05	0.84	0.84	4m	5m
userCL[1]	kmed	Euc	0.50	0.32	0.02	0.13	0.82	0.82	4.7s	39s	0.44	0.34	0.01	0.08	0.85	0.84	4m	23m
userCL[2]	kmed	Euc	0.51	0.35	0.02	0.15	0.83	0.83	4.7s	39s	0.47	0.34	0.01	0.08	0.86	0.84	4m	23m
userCL[3]	kmed	Euc	0.55	0.31	0.03	0.14	0.83	0.82	4.7s	39s	0.42	0.31	0.01	0.08	0.84	0.84	4m	23m
userkNN	–	Cos	**0.61**	**0.45**	0.03	**0.19**	0.85	0.85	–	2m	**0.63**	**0.50**	0.02	**0.13**	0.88	0.87	–	2.5h
userCL[1]	spect	Cos	0.46	0.33	0.02	0.13	0.85	0.85	4.3s	14s	0.36	0.33	8e-3	0.06	0.87	0.87	4m	5.6m
userCL[2]	spect	Cos	0.52	0.35	0.02	0.14	0.85	0.85	4.3s	14s	0.45	0.35	0.01	0.08	0.87	0.87	4m	5.6m
userCL[3]	spect	Cos	0.51	0.34	0.02	0.13	0.83	0.84	4.3s	14s	0.40	0.35	0.01	0.07	0.87	0.87	4m	5.6m
userCL[1]	kmed	Cos	0.49	0.35	0.02	0.14	0.82	0.84	4.6s	12s	0.43	0.33	0.01	0.07	0.87	0.86	4m	4.6m
userCL[2]	kmed	Cos	0.54	0.37	0.03	0.15	0.83	0.84	4.6s	12s	0.43	0.35	0.01	0.07	0.87	0.86	4m	4.6m
userCL[3]	kmed	Cos	0.52	0.36	0.02	0.15	0.84	0.84	4.6s	12s	0.40	0.35	0.01	0.07	0.86	0.86	4m	4.6m

method			BX								Last.fm							
	params		Precision		Recall		nDCG				Precision		Recall		nDCG			
	clust	*d*	@1	@10	@1	@10	@1	@10	mt	tt	@1	@10	@1	@10	@1	@10	mt	tt
userkNN	–	Euc	0.05	0.03	3e-3	0.02	0.39	0.42	–	12.5h	0.18	0.14	4e-4	3e-3	0.04	0.12	–	6m
PureCL	spect	–	0.03	0.03	2e-3	0.02	0.36	0.41	3m	4m	0.39	0.30	9e-4	0.01	0.09	0.14	1.2s	22s
userCL[1]	spect	Euc	0.07	0.04	5e-3	0.03	0.39	0.43	3m	4.3m	0.53	0.38	2e-3	0.01	**0.15**	0.17	1.5s	0.2s
userCL[2]	spect	Euc	0.07	0.04	5e-3	0.03	0.40	0.43	3m	4.3m	0.53	0.39	2e-3	0.01	**0.15**	0.16	1.5s	0.2s
userCL[3]	spect	Euc	0.07	0.04	5e-3	0.03	0.39	0.43	3m	4.3m	0.60	0.40	2e-3	0.01	0.13	0.16	1.5s	0.2s
PureCL	kmed	–	0.03	0.02	2e-3	0.01	0.35	0.41	3m	3.5m	0.30	0.29	9e-4	0.01	0.08	0.14	1.2s	20s
userCL[1]	kmed	Euc	0.06	0.03	5e-3	0.02	0.38	0.42	3m	5m	0.53	0.38	2e-3	0.01	0.13	0.16	1.2s	0.3s
userCL[2]	kmed	Euc	0.06	0.03	4e-3	0.02	0.38	0.42	3m	5m	0.50	0.36	1e-3	0.01	0.12	0.16	1.2s	0.3s
userCL[3]	kmed	Euc	0.06	0.03	5e-3	0.02	0.39	0.42	3m	5m	0.56	0.37	2e-3	0.01	0.13	0.16	1.2s	0.3s
userkNN	–	Cos	**0.13**	**0.07**	**0.01**	**0.06**	**0.44**	0.45	–	1.1h	0.41	0.36	9e-4	0.01	0.10	0.18	–	31s
userCL[1]	spect	Cos	0.05	0.03	4e-3	0.02	0.37	0.42	3m	1m	**0.62**	0.42	2e-3	0.01	**0.15**	0.17	1.2s	0.1s
userCL[2]	spect	Cos	0.09	0.05	6e-3	0.03	0.40	0.44	3m	1m	**0.62**	**0.44**	2e-3	0.01	**0.15**	0.17	1.2s	0.1s
userCL[3]	spect	Cos	0.07	0.04	5e-3	0.03	0.40	0.43	3m	1m	**0.62**	**0.44**	2e-3	0.01	**0.15**	0.17	1.2s	0.1s
userCL[1]	kmed	Cos	0.06	0.03	4e-3	0.02	0.39	0.43	3m	1m	**0.61**	0.44	2e-3	0.01	**0.15**	0.17	1.1s	0.1s
userCL[2]	kmed	Cos	0.06	0.03	4e-3	0.02	0.39	0.43	3m	1m	0.59	**0.45**	2e-3	0.01	**0.15**	0.17	1.1s	0.1s
userCL[3]	kmed	Cos	0.07	0.04	5e-3	0.02	0.40	0.43	3m	1m	0.57	0.44	2e-3	0.01	**0.15**	0.17	1.1s	0.1s

was reduced in the three datasets. Note that userCL on k-medians and spectral clustering is expected to be faster than userkNN. Thus, the cost of time saving is a reduction on performance in terms of Precision and Recall, as the last columns of these results indicate (columns depicted by mt and tt). These columns show model and testing times for each experiment. Note that the testing time for userkNN is in the order of hours and cluster-based methods takes only minutes in ML1M and BX, but at the cost of precision lost.

Results achieved on the last dataset (Last.fm) are very interesting. Table 3 shows that the gap between userkNN and PureCL or userCL is reduced. In fact, userCL outperforms userkNN in terms of Precision. Note that the Recall is extremely low in this dataset, fact that shows that the recommendation problem in Last.fm is very hard due to its sparseness. Curiously, the comparison of both algorithms in terms of nDCG shows a tie, with only 1% point of difference in terms of nDCG in favor of userkNN. This result has merits in the sense that clustering-based methods, whose purpose is to reduce the computational time involved in large scale systems, are also capable to produce recommendations of similar quality in large scale datasets. The difference in performance in this case can be explained from the unbalance between number of users and number of items, a fact that in finer-granularity systems as video recommender systems in online platforms (e.g. Youtube) or song recommendations services (e.g. Spotify or Last.fm) is common. This finding suggests that cluster-based algorithms can help in recommender systems where the number of items is very large. Last columns of last.fm experiments show model and testing times for these experiments. Whilst userkNN takes minutes in testing, cluster-based methods take only a fraction of second. This finding illustrate the potential of the proposal in terms of efficiency.

To understand our results, we show in Figure 3 the number of inside neighbors (users in the same cluster to the target user) used in neighborhood computation over spectral clustering. Each bin indicates the number of users that reach the number of inside users indicated by the bin. The figure shows that the recommendation quality is strongly related to the number of inside users. In Last.fm dataset (best performance for userCL), many neighborhood users come from inside the cluster. This fact does not occur in the rest of datasets, where the majority of the neighbors are recovered from other clusters. This result indicates that the quality of the recommendations produced by userCL strongly depends on the quality of the clusters.

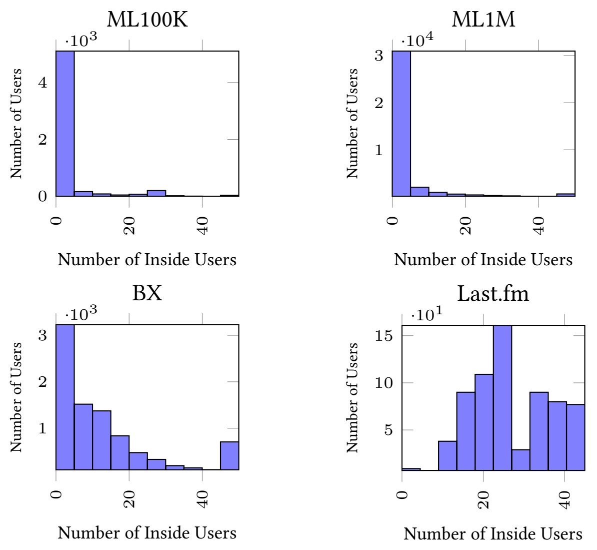

Figure 3: Histograms of Inside Users

6 CONCLUSIONS

In this paper, we proposed a clustering-based method for Top-N recommendation, userCL, which uses a novel distance function to compute the neighborhood of a given user. We conducted a comprehensive set of experiments and compared userCL with other Top-N recommendation algorithms. UserCL generates recommendations by aggregating the ratings of users that are similar to those located in the same cluster. However, userCL considers the ratings of users in different clusters, allowing to improve the quality of Top-N recommendations. Our experiments indicate that userCL reduces the gap in terms of recommendation quality with userkNN, but significantly improves the efficiency for kNN computation achieving a speed up of 10x when spectral clustering or k-medians is used.

Although k-medians or spectral clustering have achieved promising performance in recommendations in last.fm, they have limitations. These algorithms cannot prevent the arise of highly skewed clusters. Prototype-based clustering algorithms quite often generate some clusters that are extremely small. We believe that it is necessary to obtain well-formed as well as reasonably balanced clusters to improve the quality of the recommendations provided by userCL.

REFERENCES

[1] J. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *UAI '98: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, Madison, Wisconsin, USA, July 24-26*, pages 43–52, 1998.
[2] O. Celma. *Music Recommendation and Discovery in the Long Tail*. Springer, 2010.
[3] J. Das, S. Majumder, D. Dutta, and P. Gupta. *Iterative Use of Weighted Voronoi Diagrams to Improve Scalability in Recommender Systems*, pages 605–617. Springer, 2015.
[4] J. Das, P. Mukherjee, S. Majumder, and P. Gupta. Clustering-based recommender system using principles of voting theory. In *International Conference on Contemporary Computing and Informatics (IC3I)*, pages 230–235. IEEE, 2014.
[5] M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems*, 22(1):143–177, 2004.
[6] T. George and S. Merugu. A scalable collaborative filtering framework based on co-clustering. In *Data Mining, Fifth IEEE International Conference on*, pages 4–pp. IEEE, 2005.
[7] M. Hahsler. Recommenderlab: A framework for developing and testing recommendation algorithms, r package, 2016.
[8] X.-M. Jiang, W.-G. Song, and W.-G. Feng. Optimizing collaborative filtering by interpolating the individual and group behaviors. In X. Zhou, J. Li, H. T. Shen, M. Kitsuregawa, and Y. Zhang, editors, *Frontiers of WWW Research and Development - APWeb 2006: 8th Asia-Pacific Web Conference*, pages 568–578. Springer, 2006.
[9] J. Kelleher and D. Bridge. Rectree centroid: An accurate, scalable collaborative recommender. *AICS 2003*, page 7, 2003.
[10] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *CSCW '94, Proceedings of the Conference on Computer Supported Cooperative Work, Chapel Hill, NC, USA, October 22-26*, pages 175–186, 1994.
[11] B. M. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering. In *Proceedings of the fifth international conference on computer and information technology*, volume 1. Citeseer, 2002.
[12] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the Tenth International World Wide Web Conference, WWW 01, Hong Kong, China, May 1-5*, pages 285–295, 2001.
[13] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen. *Collaborative filtering recommender systems. In The adaptive web.* Springer Berlin Heidelberg, 2007.
[14] V. Strnadová-Neeley, A. Buluç, J. R. Gilbert, L. Oliker, and W. Ouyang. Lira: A new likelihood-based similarity score for collaborative filtering. In *Workshop on Large Scale Recommender Systems co-located with ACM RecSys*, 2016.
[15] G.-R. Xue, C. Lin, Q. Yang, W. Xi, H.-J. Zeng, Y. Yu, and Z. Chen. Scalable collaborative filtering using cluster-based smoothing. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, pages 114–121. ACM, 2005.