Towards a Scalable *k*NN CF Algorithm: Exploring Effective Applications of Clustering

Al Mamunur Rashid, Shyong K. Lam, Adam LaPitz, George Karypis, and John Riedl

Computer Science and Engineering
University of Minnesota
Minneapolis, MN 55455
`{arashid, lam, lapitz, karypis, riedl}@cs.umn.edu`

Abstract. Collaborative Filtering (CF)-based recommender systems bring mutual benefits to both users and the operators of the sites with too much information. Users benefit as they are able to find items of interest from an unmanageable number of available items. On the other hand, e-commerce sites that employ recommender systems can increase sales revenue in at least two ways: a) by drawing customers' attention to items that they are likely to buy, and b) by cross-selling items. However, the sheer number of customers and items typical in e-commerce systems demand specially designed CF algorithms that can gracefully cope with the vast size of the data. Many algorithms proposed thus far, where the principal concern is recommendation quality, may be too expensive to operate in a large-scale system. We propose ClustKnn, a simple and intuitive algorithm that is well suited for large data sets. The method first compresses data tremendously by building a straightforward but efficient clustering model. Recommendations are then generated quickly by using a simple Nearest Neighbor-based approach. We demonstrate the feasibility of ClustKnn both analytically and empirically. We also show, by comparing with a number of other popular CF algorithms that, apart from being highly scalable and intuitive, ClustKnn provides very good recommendation accuracy as well.

1 Introduction

The amount of content available on the web today is tremendous. The English version of the online encyclopedia Wikipedia contains more than 1.1 million articles. Flickr, a popular photo sharing service, has about 130 million photos[1]. The blog search engine Technorati has over 41 million blogs and 2.5 billion links in its index. This is far too much content for any person to consume, and is, in a nutshell, the problem of *information overload*. To help solve this problem, people

[1] *http://time.com/time/magazine/article/ 0,9171,1186931,00.html*

need tools to help them decide what items might be worthwhile to look at. One effective tool for this task is a *recommender system.* These systems suggest items that a user might be interested based on her preferences, observed behaviors, and information about the items themselves.

An example of a recommender system in use is the personalized internet radio station last.fm[2], which chooses songs to play for a user based on the songs and artists that she has listened to and expressed opinions about in the past. Another example is MovieLens[3], a movie recommender that uses peoples' opinions about movies to recommend other movies that users might enjoy watching.

Collaborative Filtering. Recommender systems are often implemented using an *automated collaborative filtering* (ACF, or CF) algorithm. These algorithms produce recommendations based on the intuition that similar users have similar tastes. That is, people who you share common likes and dislikes with are likely to be a good source for recommendations. Numerous CF algorithms have been developed over the past fifteen years, each of which approach the problem from a different angle, including similarity between users[20], similarity between items[23], personality diagnosis[19], Bayesian networks[2], singular value decomposition[25], and neural networks[18]. These algorithms have distinguishing qualities with respect to evaluation metrics such as recommendation accuracy, speed, and level of personalization.

When deciding which algorithm to use in a system, one key factor to consider is the algorithm's ability to scale given the size of the data. In systems with millions of items and possibly tens of millions of users, the number of CF algorithms that are practically able to produce quality recommendations in real time is limited. Even with costs of commodity hardware falling rapidly, a brute-force approach may be prohibitively expensive. Tradeoffs between speed and recommendation accuracy often need to be made, and the problem of developing highly scalable algorithms continues to be an interesting problem.

Efficient and Scalable CF Algorithms. Yu et al. note in [32] that there has been relatively little work in studying the efficiency of CF algorithms and developing algorithms that do not have either extremely expensive precomputation time or slow online performance. Linden et al. explore the suitability of several algorithms for use on the Amazon.com web site and conclude that algorithms based on similarity between items are the best choice for a system of their size[14]. They consider algorithms based on clustering techniques, but dismiss those algorithms on the premise that they produce poor recommendation quality. However, other researchers have found that using clustering techniques can indeed lead to good recommendations[4, 30, 22, 13]. The algorithm proposed in this paper is based on classical clustering methods, and based on our results, we also believe that using clustering is a viable way to increase efficiency and scalability while maintaining good recommendation quality. A more in-depth summary of previous work in applying clustering methods to collaborative filtering can be found in [13].

[2] *http://last.fm*
[3] *http://www.movielens.umn.edu*

Contributions. In this paper, we propose ClustKnn, a hybrid memory and model-based CF algorithm based on clustering techniques, as a way to overcome this scalability challenge. By applying complexity analysis, we analyically demonstrate the performance advantages that ClustKnn has over traditional CF algorithms. In addition, we present empirical measurements of the performance and recommendation accuracy of ClustKnn and several other algorithms.

The remainder of this paper is organized as follows. Section 2 introduces the general framework in which CF algorithms operate in, and further discusses the problem that we are solving. Section 3 describes our proposed approach in detail. Section 4 outlines several other well-known CF algorithms that we compare our approach to. The results of our comparison are presented in section 5 and discussed in section 6. Finally, we conclude in section 7 with a brief discussion of future work.

2 The Problem Domain

A collaborative filtering domain consists of a set of n customers or users $\{u_1, u_2, \ldots, u_n\}$, a set of m products or items $\{a_1, a_2, \ldots, a_m\}$, and users' preferences on items. Typically, each user only expresses her preferences for a small number of items. In other words, the corresponding $user \times item$ matrix is very sparse.

Users' preferences can be in terms of *explicit* ratings on some scale including a binary like/dislike, or they can be *implicit*—for example, a customer's purchase history, or her browsing patterns. A recommender system may also maintain demographic and other information about the users, and information about item features such as actors, directors, and genres in the case of a movie. This additional content information can be used to create *content-based filtering* [16, 21], which can help improve a CF system, particularly where rating data is limited or absent (e.g., newly introduced items). In this paper we consider CF systems consisting of explicit numerical ratings and no content information.

Next we address two semantically different types of recommendations. A CF recommender system can produce two forms of recommendations on the items the target user has not already rated: a) predicted ratings on the items, and b) an ordered list of items the user might like the most. The latter type of recommendations is sometimes referred to as *top-N* recommendations [25, 23]. Note that a *top-N* list can be trivially constructed by first computing rating predictions on all items not yet rated, and then sorting the result and keeping the top N. We study both types of recommendation in this paper.

We now turn to the problem statement. An e-commerce recommender system may easily involve millions of customers and products [14]. This amount of data poses a great challenge to the CF algorithms in that the recommendations need to be generated in real-time. Furthermore, the algorithm also has to cope with a steady influx of new users *and* items. For the majority of the algorithms proposed to date, the primary emphasis has been given into improving recommendation accuracy. While accuracy is certainly important and can affect the profitability of the company, the operator simply cannot deploy the system if it does not scale to the vast data of the site.

3 Proposed Approach

In [2], Breese et al. introduce a classification of CF algorithms that divides them into two broad classes: *memory-based* algorithms and *model-based* algorithms. Here, we briefly discuss each of these and describe how our approach leverages the advantages of both types of algorithms.

A memory-based algorithm such as User-based KNN [20] utilizes the entire database of user preferences when computing recommendations. These algorithms tend to be simple to implement and require little to no training cost. They can also easily take new preference data into account. However, their online performance tends to be slow as the size of the user and item sets grow, which makes these algorithms as stated in the literature unsuitable in large systems. One workaround is to only consider a subset of the preference data in the calculation, but doing this can reduce both recommendation quality and the number of items that can be recommended due to data being omitted from the calculation. Another workaround is to perform as much of the computation as possible in an offline setting. However, this may make it difficult to add new users to the system on a real-time basis, which is a basic necessity of most online systems. Furthermore, the storage requirements for the pre-computed data could be high.

On the other hand, a model-based algorithm such as one based on Bayesian networks [2] or singular value decomposition (SVD) [25] computes a model of the preference data and uses it to produce recommendations. Often, the model-building process is time-consuming and is only done periodically. The models are compact and can generate recommendations very quickly. The disadvantage to model-based algorithms is that adding new users, items, or preferences can be tantamount to recomputing the entire model.

ClustKnn, our proposed approach is a hybrid of the *model* and *memory* based approaches and has the advantages from both types. One of our primary goals is to maintain simplicity and intuitiveness throughout the approach. We believe this is important in a recommender algorithm because the ability to succintly explain to users how recommendations are made is a major factor in providing a good user experience [29]. We achieve this by utilizing a straightforward *partitional clustering* algorithm [12] for modeling users. To generate recommendations from the learned model, we use a nearest-neighbor algorithm similar to the one described in [20]. However, since the data is greatly compressed after the model is built, recommendations can be computed quickly, which solves the scalability challenge discussed previously.

One interesting property of ClustKnn is its tunable nature. We show later in the paper that a tunable parameter, the number of clusters k in the model, can be adjusted to trade off accuracy for time and space requirements. This makes ClustKnn adaptable to systems of different sizes and allows it to be useful throughout the life of a system as it grows.

We now provide the details of the algorithm. First we give an outline, and following that we provide explanations of the key points. The algorithm has two phases: model building (offline) and generation of predictions or recommendations (online).

Model Building

- Select the number of user-clusters k, considering the effect on the recommendation accuracy and resource requirements.
- Perform BISECTING k-MEANS clustering on the user-preference data.
- Build the model with k *surrogate* users, directly derived from the k *centroids*: $\{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_k\}$, where each $\mathbf{c}_i$ is a vector of size m, the number of items. That is, $\mathbf{c}_i = (\tilde{R}_{c_i,a_1}, \tilde{R}_{c_i,a_2}, \ldots, \tilde{R}_{c_i,a_m})$, where $\tilde{R}_{c_i,a_j}$ is the element in the centroid vector $\mathbf{c}_i$ corresponding to the item a_j. Further, since $\tilde{R}_{c_i,a_j}$ is essentially an average value, it is 0 if nobody in the i-th cluster has rated a_j.

Prediction Generation

In order to compute the rating prediction $\hat{R}_{u_t,a_t}$ for the target (user, item) pair (u_t, a_t), the following steps are taken.

- Compute similarity of the target user with each of the surrogate model users who have rated a_t using the Pearson correlation coefficient:

$$w_{u_t,c_i} = \frac{\sum_{a\in\mathcal{I}}(R_{u_t,a} - \overline{R}_{u_t})(\tilde{R}_{c_i,a} - \overline{R}_{c_i})}{\sqrt{\sum_{a\in\mathcal{I}}(R_{u_t,a} - \overline{R}_{u_t})^2 \sum_{a\in I}(\tilde{R}_{c_i,a} - \overline{R}_{c_i})^2}}$$

  where $\mathcal{I}$ is the set of items rated by both the target user and i-th surrogate user.
- Find up to l surrogate users most similar to the target user.
- Compute prediction using the adjusted weighted average:

$$\hat{R}_{u_t,a_t} = \overline{R}_{u_t} + \frac{\sum_{i=1}^{l}(\tilde{R}_{c_i,a_t} - \overline{R}_{c_i})w_{u_t,c_i}}{\sum_{i=1}^{l} w_{u_t,c_i}}$$

Note that any *partitional clustering* [12] technique can used for model-building in CLUSTKNN. We selected the BISECTING k-MEANS algorithm, which we describe below.

BISECTING k-MEANS is an extension to and an improved version of the basic k-MEANS algorithm [12]. The algorithm starts by considering all data points (rating-profiles of all users, in our case) as a single cluster. Then it repeats the following steps $(k-1)$ times to produce k clusters.

1. Pick the largest cluster to split.
2. Apply the basic k-MEANS (2-MEANS, to be exact) clustering to produce 2 sub-clusters.
3. Repeat step 2 for j times and take the best split, one way of determining which is looking for the best *intra*-cluster similarity.

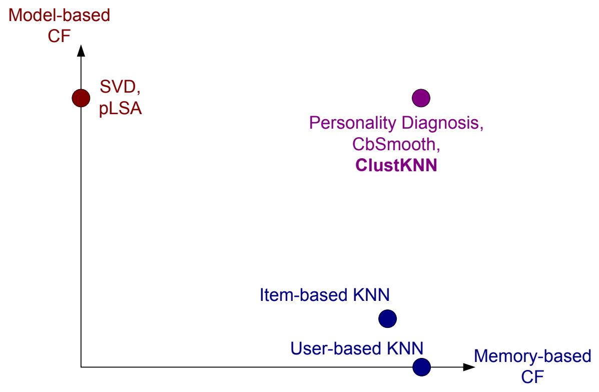


Fig. 1. The space encompassed by the CF algorithms we studied.

At this stage, it is straightforward to derive the time-complexity of ClustKnn. Note that the time complexity of CF algorithms can be divided into two parts: one for the offline model-building, and the other for the online generation of recommendations.

The time-complexity of the basic k-means is reported to be $O(n)$ in [12]; however, this is assuming the cost of computing the *similarity* or *distance* between the data points and centroids as a constant. However, in ClustKnn, this cost is $O(m)$, so the k-means time-complexity becomes $O(mn)$. Therefore, the complexity of the Bisecting k-means becomes $O((k-1)jmn) \simeq O(mn)$, which is the offline complexity of ClustKnn.

During the online stage, $O(k)$ similarity weight calculations are needed for the target user, each of which takes $O(m)$ time; therefore, online time-complexity is $O(km) \simeq O(m)$.

In their work on document clustering [28], Steinbach et al. empirically showed that Bisecting k-means performed the best on a set of text datasets. Furthermore, the authors noted a nice property of Bisecting k-means—the produced clusters tended to be of relatively uniform size. Whereas, in regular k-means, the cluster sizes may vary significantly, producing poorer quality clusters.

4 Other CF algorithms Considered

Xue et al [31] took the same idea of using clustering to transform a k-NN-based CF algorithm scalable. We study this algorithm side by side with our approach. In order to investigate how ClustKnn compares with other CF algorithms, we selected several other algorithms shown in figure 1. Our criteria for picking the algorithms include a) how frequently the algorithms are cited in the literature, and b) whether the algorithms span the classification space introduced by Breese et al [2]. In the following, we provide a brief overview of each of the selected algorithms.

pLSA
Probabilistic Latent Semantic Analysis (pLSA) for collaborative filtering is an elegant *generative* model proposed by Hofmann et al [11]. pLSA is a *three-way aspect* model adapted from their earlier contribution of *two-way aspect* models applied to text analysis [10].

At the heart of the pLSA approach is the notion of the *latent* class variable Z. The number of states of Z is an input to the model, and each state z can be interpreted as a different *user-type*. Each user belongs to these user-types with a unique probability distribution $P(z|u)$. Recall that this type of probabilistic assignment of entities to groups is similar in principle to the so-called *soft-clustering* approach.

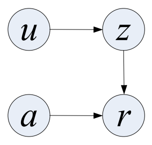


Fig. 2. 3-way aspect model.

Hofmann models the probability density function $p(r|a,z)$ with a Gaussian mixture model and develops an Expectation Maximization (EM) method to learn mixture coefficients $P(z|u)$ and $p(r|a,z)$. Note that, due to Gaussian modeling, estimating $p(r|a,z)$ becomes estimating $p(r;\mu_{a,z},\sigma_{a,z})$.

In the end, the learned model includes $P(z|u)$s for each user and for each state of Z, and values of μ and σ for each item and each state of Z.

Prediction for the target (user, item) pair is simply the weighted average of the means of a_t for each state z. That is,

$$\hat{R}_{u_t,a_t} = \sum_z P(z|u_t)\mu_{a_t,z} \tag{1}$$

Note that the model size grows linearly with the number of users; in fact, it is $O(m+n) \simeq O(n)$, if $n \gg m$. Furthermore, since $P(z|u)$'s are precomputed in the model, recommending to the new users pose a challenge. Hofmann proposes to perform a limited EM iteration in this situation.

SVD
Singular Value Decomposition (SVD) is a matrix factorization technique that can produce three matrices given the rating matrix A: $SVD(A) = U \times S \times V^T$. Details of SVD can be found in [6]; however, suffice it to say that the matrices U, S, and V can be reduced to construct a *rank-k* matrix, $X = U_k \times S_k \times V_k^T$ that is the closest approximation to the original matrix.

SVD requires a complete matrix to operate; however, a typical CF rating matrix is very sparse (see table 2). To circumvent this limitation of the CF datasets, [25] proposed using average values in the empty cells of the rating matrix. An alternate method proposed by Srebro et al. [27] finds a model that maximizes the *log-likelihood* of the actual ratings by an EM procedure. The EM procedure is rather simple and is stated below:

E-step: Missing entries of A are replaced with the values of current X. This creates an *expected* complete matrix A'.

M-step: Perform $SVD(A')$. This creates un updated X.

This EM process is guaranteed to converge. Upon convergence, the final X represents a linear model of the rating data, and the missing entries of the original A are filled with predicted values.

Personality Diagnosis

Personality Diagnosis [19] is a probabilistic CF algorithm that lies in between *model-based* and *memory-based* approaches. In this CF algorithm, each user is assumed to have a *personality type* that captures their true, internal preferences for items. However, the true personality type is unobservable, since users rate items by adding a *Gaussian* noise to their true preferences on the items.

The probability that the target user u_t's rating on an item a_t is x, given u_t and u_i's personality types are same, is defined by equation 2.

$$P(R_{u_t,a_t} = x | type_{u_t} = type_{u_i}) = e^{-(x - R_{u_i,a_t})^2/2\sigma^2} \tag{2}$$

The authors derive the probability that two users' personalities are of the same type as follows.

$$P(type_{u_t} = type_{u_i} | \mathbf{R}_{u_t}) = 1/n \prod_{a \in \mathcal{I}} P(R_{u_t,a} = x_a | type_{u_t} = type_{u_i}) \tag{3}$$

where $\mathbf{R}_{u_t}$ is the set of ratings reported by the target user.

Finally, the prediction on the target item a_t for u_t is computed as

$$\begin{aligned} \hat{R}_{u_t,a_t} &= \underset{x}{\operatorname{argmax}}\, P(R_{u_t,a_t} = x | \mathbf{R}_{u_t}) \qquad (4) \\ &= \underset{x}{\operatorname{argmax}} \sum_i P(R_{u_t,a_t} = x | type_{u_t} = type_{u_i}) \\ &\qquad .P(type_{u_t} = type_{u_i} | \mathbf{R}_{u_t}) \qquad (5) \end{aligned}$$

User-based KNN

This algorithm belongs to the *memory-based* class of CF algorithms. Predictions under this algorithm are computed as a two step process. First, the similarities between the target user and all other users who have rated the target item are computed — most commonly using the Pearson correlation coefficient [8, 20]. That is,

$$w_{u_i u_t} = \frac{\sum_{a \in \mathcal{I}} (R_{u_i,a} - \overline{R}_{u_i})(R_{u_t,a} - \overline{R}_{u_t})}{\sqrt{\sum_{a \in \mathcal{I}} (R_{u_i,a} - \overline{R}_{u_i})^2 \sum_{a \in I} (R_{u_t,a} - \overline{R}_{u_t})^2}} \tag{6}$$

where $\mathcal{I}$ is the set of items rated by both of the users.

Then the prediction for the target item a_t is computed using at most k closest users found from step one, and by applying a weighted average of deviations from the selected users' means:

$$\hat{R}_{u_t,a_t} = \overline{R}_{u_t} + \frac{\sum_{i=1}^{k}(R_{u_i,a_t} - \overline{R}_{u_i})w_{u_i,u_t}}{\sum_{i=1}^{k} w_{u_i,u_t}} \tag{7}$$

Note that we follow a number of improvements suggested in [8], including dividing similarities by a constant if the two users have not co-rated enough items.

Item-based KNN
This algorithm is also an instance of a *memory-based* approach. Predictions are computed by first computing item-item similarities. [23] proposed adjusted cosine measure for estimating the similarity between two items a, and b:

$$w_{a,b} = \frac{\sum_{u_i \in \mathcal{U}}(R_{u_i,a} - \overline{R}_{u_i})(R_{u_i,b} - \overline{R}_{u_i})}{\sqrt{\sum_{u_i \in \mathcal{U}}(R_{u_i,a} - \overline{R}_{u_i})^2 \sum_{u_i \in \mathcal{U}}(R_{u_i,b} - \overline{R}_{u_i})^2}} \tag{8}$$

Where, $\mathcal{U}$ denotes the set of users who have rated both a and b.

Once the *item-item* similarities are computed, the rating space of the target user u_t is examined to find all the rated items similar to the target item a_t. Then equation 9 is used to perform the weighted average that generates the prediction. Typically, a threshold of k similar items are used rather than all.

$$\hat{R}_{u_t,a_t} = \frac{\sum_{all\_similar\_items,d}(w_{a_t,d} * R_{u_t,d})}{\sum_{all\_similar\_items,d}(|w_{a_t,d}|)} \tag{9}$$

CbSmooth
In their paper [31], the authors present a framework to address two issues of the recommender systems: rating sparseness and algorithm scalability. We briefly discuss the framework as the following steps:

- Step 1: Cluster users into a pre-determined number of groups. Authors use the k-means clustering algorithm and Pearson correlation coefficient as the similarity function.
- Step 2: Replace the missing ratings of each user using the cluster a user belongs to. If a user u_t has not rated an item a, a *smoothed* rating is injected as a combination of the average rating of u_t and the average deviation of ratings on a by all users in u_t's cluster who rated a. That is, $\overline{R}_{u_t} + \sum_{u_i \in \mathcal{C}(u_t,a)}(R_{u_i,a} - \overline{R}_{u_i})/|\mathcal{C}(u_t,a)|$, where $\mathcal{C}(u_t,a)$ indicates all the users of u_t's cluster who rated a.
- Step 3: Find the most similar clusters of each user by computing the similarity between a user and the centroids of the clusters. Pre-select the users of the closest clusters of the target user so that neighbors are sought only from these pre-selected users.
- Step 4: Recompute the similarity between the active user and the pre-selected users by weighting each pre-selected user's ratings based on whether the

Table 1. Comparison of time-complexities of the selected CF algorithms.

CF algorithm	**Offline**	**Online**	**Scalable?**
pLSA	$O(mn)$	$O(m)$	Yes
SVD	$O(n^2m + m^2n)$	$O(m)$	Yes, expensive offline
Personality Diagnosis	-	$O(mn)$	No
ClustKnn	$\boldsymbol{O(mn)}$	$\boldsymbol{O(m)}$	Yes
User-based KNN	-	$O(mn)$	No
Item-based KNN	-	$O(mn)$	No; yes with precomputation
CbSmooth	$O(mn)$	$O(mn)$ or $O(m)$	Yes; needs $O(n^2)$ memory

rating is actual or *smoothed*. A parameter λ, where $0 \leq \lambda \leq 1$, is used to provide a weight of $(1 - \lambda)$ for an actual rating, and a weight of λ for a smoothed rating. Step 5: Let us denote these weights by w_λ.

– Compute recommendations for a user by selecting top K most similar users found in step 4, and by applying an equation similar to 7, however, incorporate w_λs for neighbors' ratings.

Authors say that a number of different algorithms can emanate from this framework by including or not including some of the steps, such as smoothing missing rating and neighbor pre-selection. However, their flagship algorithm is what authors call Scbpcc, which includes all of the steps above. We denote this algorithm by CbSmooth (Cluster-based Smoothing) in this paper.

CbSmooth addresses the rating sparsity problem by introducing smoothed ratings (step 2 above). However, CbSmooth is not scalable. Authors advocate to pre-select about 30% of all users for each user. This means that the computations still remain $O(mn)$ and are reduced by a constant factor only. CbSmooth can be made scalable if the neighbor pre-selections are performed during the offline phase. However, this imposes an additional memory requirement of $O(n^2)$, which can be prohibitive in many systems.

Comparison of time-complexity

Table 1 shows the time complexities of all the CF algorithms we address in this paper including ClustKnn. We have collected the complexity-values directly from the respective papers where they were introduced, without formally deriving them here. We, however, translate the values into the notations we follow in this paper. For an example, Hofmann [11] shows that the offline time complexity of pLSA is $O(kN)$, where k is the number of states of Z and N is the total number of ratings in the system. Since in the worst case, $N = nm$, we use the offline complexity to be $O(mn)$.

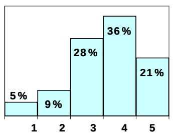
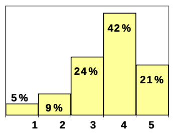
From the table it is clear that ClustKnn is one of the cheapest CF algorithms presented, considering both the offline and online time complexities. Since ClustKnn produces recommendations for a user in $O(m)$ time, it effectively transforms the User-based KNN into a highly scalable version by reducing the online time from $O(mn)$ to linear in m. Note that although the time complexities of pLSA and ClustKnn are identical, ClustKnn is much simpler and operates on an intuitive basis.

5 Empirical Analysis

5.1 Datasets

We derived our datasets from MovieLens, a research recommender site maintained by the GroupLens project[4]. Although the registered users of MovieLens can perform activities like adding tags, adding and editing movie-information, engaging in forum discussions, and so forth, the main activity taking place is rating movies so that they can receive personalized movie recommendations. As of this writing, MovieLens has more than 105,000 registered members, about 9,000 movies, and more than 13 million ratings.

Table 2. Properties of the datasets

Property	Ml1m	MlCurrent
Number of users	6,040	21,526
Number of movies	3,706	8,848
Number of ratings	10,00,209	29,33,690
Minimum $\vert u_i \vert$, $\forall i$	20	15
Average rating	3.58	3.43
Sparsity	95.5%	98.5%
Rating distribution		

We use two datasets in this paper. The first dataset is publicly available. The second dataset has been created by taking the latest 3 million ratings and the corresponding users and movies. We denote the former dataset as Ml1m and the latter as MlCurrent throughout the paper. Table 2 summarizes the number of users, number of movies, number of ratings, minimum number of ratings of each user, *sparsity*, and rating distribution of each dataset. Sparsity of a dataset is defined as the percent of empty cells (that is, no rating) in the *user* $\times$ *movie* matrix.

One key difference between the two datasets is in the rating scale. In Ml1m, the rating scale is 1 star to 5 stars, with an increment of 1 star; however, for the last couple of years MovieLens has enabled half-star ratings. As a result, in MlCurrent, the rating scale is 0.5 star to 5.0 stars, in 0.5 star increments.

[4] *http://www.cs.umn.edu/Research/GroupLens/*

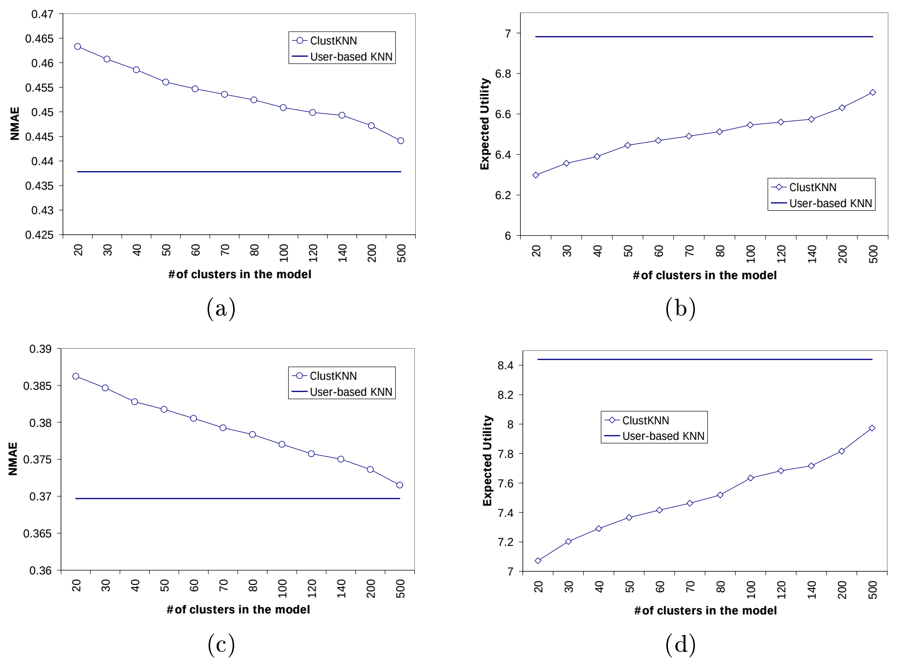


Fig. 3. Prediction performance of ClustKnn: (a)-(b) on Ml1m, and (c)-(d) on Ml-Current dataset. Results of user-based KNN are shown for comparison.

Furthermore, note from the average ratings and the rating distributions that, the distributions are skewed toward higher rating values. This is perhaps a common phenomenon since people typically consume products they think they might like. Therefore, their reports on products (movies, in this context) are mostly on what they enjoyed. Another reason for positive skewness might be the user interface itself—if the products presented to the users are ordered by the likelihood that the users would like them, they may only focus on these products when submitting ratings.

5.2 Evaluation Metrics

In this section we briefly review the metrics we use to evaluate the quality of recommendations produced by the CF algorithms. The first two to follow are to evaluate rating-predictions, and the last category is to evaluate top-N recommendations.

NMAE

Mean Absolute Error (MAE) is the most commonly applied evaluation metric for CF rating predictions. MAE is simply the average of the absolute deviation of the computed predictions from the corresponding actual ratings. Formally,

$$MAE = \frac{1}{N}\sum_{j=1}^{N}|R_{u_j} - \hat{R}_{u_j}| \quad (10)$$

where N represents the total number of predictions computed for all users.

According to this metric, a better CF algorithm has a lower MAE.

Other similar metrics such as Mean Squared Error (MSE) or Root Mean Squared Error (RMSE) are sometimes used for CF evaluation as well. Here, we only report MAE, as one general result from past work is that most evaluation metrics correlate well [25, 9].

In [7], the authors wondered about how good the CF algorithm MAEs are over purely random guessing. They proposed using the Normalized Mean Absolute Error (NMAE) that is computed by dividing the MAE of a CF algorithm with the expected MAE from random guessing. In this paper, we use the version of NMAE proposed in [15]. Formally,

$$NMAE = MAE/E[MAE] \quad (11)$$

Since the ML1M dataset has a rating scale of 1-5, $E[MAE]$ $= \frac{1}{25}\sum_{i=1}^{5}\sum_{j=1}^{5}|i-j| = 1.6$, assuming both ratings and predictions are generated by a uniform distribution. Similarly, for the MLCURRENT dataset, $E[MAE] = 1.65$.

Note that an NMAE value less than 1.0 means the approach is working better than random. An added benefit of using NMAE is that evaluation of CF datasets of different rating scales become comparable.

Expected Utility (EU)

A limitation of MAE is that it treats the same values of error equally across the space of the rating scale. For example, MAE would find no difference between the two $(\hat{R}, R)$ pairs $(5.0, 2.0)$ and $(2.0, 5.0)$. However, depending on the underlying product-domain, the users may be unhappy more about the former pair than the latter.

In order to overcome this limitation, we propose the Expected Utility (EU) metric, a variant of which can be commonly found in *Decision Theory*.

For this accuracy metric, we arrange a 10×10 matrix for a CF algorithm, where rows represent predictions, and the columns represent actual ratings. The (i, j)-th cell of this matrix gives the count of occurrence of the pair $(\hat{R}_i, R_j)$. We also construct a static 10×10 utility table where each entry corresponding to $(\hat{R}_i, R_j)$ is computed using the following utility formula: $U(\hat{R}_i, R_j) = R_j - 2|\hat{R}_i - R_j|$. Notice that the utility equation tries to penalize *false positives* more than *false negatives*. For example, $U(\hat{R}_i = 5, R_j = 2) = -4$, $U(\hat{R}_i = 2, R_j = 5) = -1$, $U(\hat{R}_i = 5, R_j = 5) = 5$, and $U(\hat{R}_i = 1, R_j = 1) = 1$. The interpretation is that not seeing a movie you would not like is no cost or value, not seeing a movie you would have liked is low cost (because there are many other good movies to see), seeing a movie you did not like is expensive and a waste of time, and seeing a movie you like is a good experience.

Based on these two matrices, the expected utility is computed as follows:

$$EU = \sum_{\substack{1\leq i\leq 10\\ 1\leq j\leq 10}} U(\hat{R}_i, R_j)P(\hat{R}_i|R_j) \tag{12}$$

Note that many cells of the 10×10 matrix are zeros or contain very small values; therefore, we estimate probabilities using an an m-estimate [3] smoothing. The m-estimate can be expressed as the following:

$$p = \frac{r + m * P}{n + m} \tag{13}$$

where n is the total number of examples, r is the number of times the event we are estimating the probability for occurs, m is a constant, and P is the prior probability. We have used $m = 2$ for our calculations.

Note that according to EU, the higher the EU of a CF algorithm, the better the performance is.

Precision-Recall-F1

Precision and recall [5] have been in use to evaluate information retrieval systems for many years. Mapping into recommender system parlance, precision and recall have the following definitions regarding the evaluation of top-N recommendations. Precision is the fraction of the top-N recommended items that are *relevant*. Recall is the fraction of the *relevant* items that are recommended. A third metric, F1, is the harmonic mean of precision and recall, and combines precision and recall into a single metric. Formally,

$$F1 = \frac{2 * precision * recall}{(precision + recall)} \tag{14}$$

Since the metrics involve the notion of relevancy, it is important to define what the relevant items are to a user. Furthermore, it is safe to say that users almost never enter preference information into the system on all the *relevant* items they have ever consumed—making the recall measure questionable in the CF domain. A good source of discussion on these and other CF evaluation metrics can be found in [9].

Researchers have tried a variety of ways to incorporate precision and recall into CF evaluation [1, 24]. In this paper, we follow an approach similar to Basu et al [1]. In particular, for our datasets, we consider the target user's *relevant* items known to us as the ones she rated 4.0 or above. Furthermore, since our experiment protocol involves dividing the data into training and test sets, we focus on the test set to find the actual *relevant* items of the target user and to compute the top-N list for her. Specifically, the top-N list only contains items that are in the target user's test set. Similarly, a list of *relevant* items are also constructed for the target user from her test set items. Based on the *relevant* list of and the top-N list for the target user, the usual precision-recall-F1 computation ensues.

5.3 Results

Table 3. Comparison of rating-prediction quality of the selected CF algorithms. (The best results in each column and the results of ClustKnn are shown in bold face.)

CF algorithm	MAE		NMAE		EU	
	Ml1m	MlCurrent	Ml1m	MlCurrent	Ml1m	MlCurrent
SVD	**0.69**	-	**0.43**	-	6.81	-
User-based KNN	0.70	0.61	0.44	0.37	**6.98**	8.44
Item-based KNN	0.70	**0.60**	0.44	**0.36**	6.93	**8.48**
ClustKnn (k=200)	**0.72**	**0.62**	**0.45**	**0.37**	**6.63**	**7.82**
pLSA	0.72	0.61	0.45	0.37	6.57	7.95
Personality Diagnosis	0.77	0.66	0.48	0.40	5.00	3.19
CbSmooth	0.71	0.62	0.44	0.37	6.86	8.24

Most of our empirical investigation involves taking a five-fold cross-validation approach over each dataset. In other words, we randomly partition our data into five disjoint folds and apply four folds together to train a CF algorithm, and use the remaining fold as a test set to evaluate the performance. We repeat this process five times for each dataset so that each fold is used as a test set once. The results we present are averages over five folds.

First we demonstrate the rating-prediction performance of ClustKnn. Figure 3 plots the predictive performance of ClustKnn both for the metrics NMAE and EU, and for both of the datasets. Since ClustKnn can be regarded as approximating user-based KNN with the two becoming equivalent when k equals the number of users in the system (assuming non-empty clusters), we have also included the predictive performance of user-based KNN in the plots — to consider it as an upper bound for ClustKnn. As depicted in figure 3, the performance of ClustKnn with a moderate value of k, both by MAE and EU, is nearly as good as the user-based KNN. For example, on the MlCurrent dataset, which has more than 21,500 users, a ClustKnn model with 200 clusters gives an NMAE of 0.37 and EU=7.82 — very close to the corresponding user-based KNN results: NMAE=0.36 and EU=8.44. Furthermore, a trend evident from the graph is that as k gets higher, accuracy keeps improving.

Table 3 compares prediction qualities of the ratings produced by the selected CF algorithms. Note that each algorithm requires a few parameters to be set which can be crucial for its better performance. For example, number of z in pLSA, σ in personality diagnosis, and so forth. We followed the suggestions and specifications found in the respective papers to tune the algorithms so that they perform their best.

We see from table 3 that SVD produced the best quality rating-predictions according to both NMAE and EU on the Ml1m dataset. We did not have enough computational resources available to run our particular Matlab implementation of SVD on the MlCurrent dataset. User and item-based KNN produce the

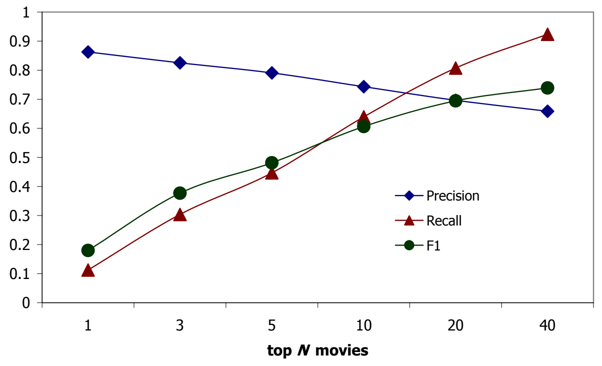


Fig. 4. ClustKnn ($k = 200$) top-N recommendation performance on Ml1m dataset with varying values of N.

next best quality predictions. ClustKnn with k=200 performs very well, and it is at least as accurate as pLSA and CbSmooth, and much better than the other hybrid model- and memory-based CF algorithm, personality diagnosis. Interestingly, paying a close attention to the NMAE and EU columns, the finding of [25, 9] that CF evaluation metrics correlate, becomes evident. Indeed, the correlation coefficient between MAE and EU on Ml1m dataset is -0.94, and on MlCurrent dataset it is -0.97. Note that negative correlations are due to the fact that the directions of MAE and EU are opposite, i.e., MAE is an *error* metric and EU is a *value* metric. Next we turn into top-N recommendation results.

Figure 4 shows the interplay between precision and recall, and the resulting F1 for ClustKnn as N varies. The pattern present in the figure is consistent across each of the CF algorithms we studied. Note that more than 50% of the users have only 12 or fewer *relevant* items in the test sets of Ml1m, and 6 or fewer in the test sets of MlCurrent. Therefore, recall values quickly ramp up and higher values of N provide less valuable information if we want to compare the algorithms.

Table 4 shows the comparative top-N recommendation results of the algorithms for N=3 and 10. The results closely follow the results in the rating predictions. Further, ClustKnn displays good top-N performance, as good as pLSA and CbSmooth, and much better than personality diagnosis.

6 Discussion

Scalability and other features. From the discussion thus far we see that ClustKnn is highly scalable—the online prediction time varies linearly with the number of items in the system. ClustKnn is *intuitive*. Its foundation rests on a neighborhood-based algorithm that embraces the *collaborative filtering* philosophy, i.e., recommend based on the neighbors' opinions. However, since there

Table 4. Comparison of top-N recommendation quality of the selected CF algorithms.

	top-3			
CF algorithm	Precision		F1	
	ML1M	MLCURRENT	ML1M	MLCURRENT
SVD	**0.8399**	-	**0.379**	-
User-based KNN	0.833	**0.6693**	0.379	**0.4086**
Item-based KNN	0.819	0.657	0.374	0.407
CLUSTKNN (k=200)	**0.825**	**0.659**	**0.377**	**0.407**
pLSA	0.817	0.656	0.375	0.406
Personality Diagnosis	0.789	0.622	0.366	0.391
CBSMOOTH	0.816	0.645	0.372	0.399

	top-10			
CF algorithm	Precision		F1	
	ML1M	MLCURRENT	ML1M	MLCURRENT
SVD	**0.7564**	-	**0.6131**	-
User-based KNN	0.750	**0.5953**	0.610	0.556
Item-based KNN	0.749	0.592	0.610	0.556
CLUSTKNN (k=200)	**0.743**	**0.589**	**0.606**	**0.553**
pLSA	0.739	0.587	0.604	0.552
Personality Diagnosis	0.723	0.565	0.595	0.537
CBSMOOTH	0.742	0.584	0.605	0.549

can be far too many users from which to find neighbors, CLUSTKNN creates a constant number of pseudo users by grouping real users. The accuracy of this hybrid *memory* and *model*-based algorithm is very good—the best algorithm in our collection is better by only a tiny percentage. The sensitivity of recommender system users to changes in algorithm accuracy has not been studied, but it is reasonably unlikely that users will notice an MAE change of less than 1%. The learned cluster model can be used to find various customer segments and their general characteristics.

Memory Footprint. The memory footprint of this algorithm is very small once the model is learned. The Memory required to generate recommendations for the target user is only $O(km + m)$, where m is the number of items in the system—$O(km)$ for the model and $O(m)$ to store the target user's profile. As a result, this algorithm is ideal for platforms with low storage and processing capabilities.

Recommendations on Handheld Devices. One such platform is handheld computers. These devices are far slower and can store much less data than their desktop counterparts. Furthermore, many devices in use today are not continuously connected to networks. Deployment of recommender systems on handheld devices is an active area of research today [17], and CLUSTKNN provides one possible way to implement a self-contained recommender system on a handheld device. CLUSTKNN can also be useful in high-usage systems where recommendation throughput is an important factor.

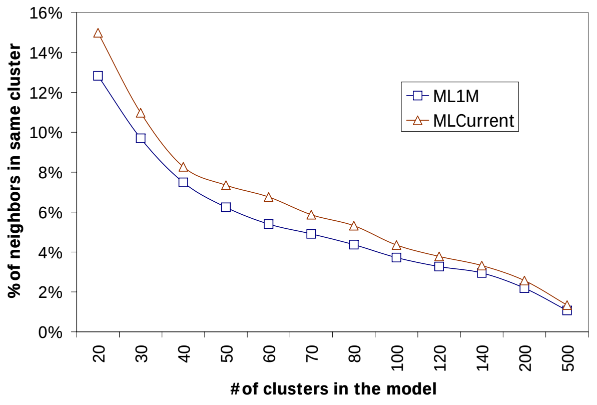


Fig. 5. Percent of top 20 neighbors that can be found in the same cluster.

Finally, we conclude our discussion with an alternate approach we could have taken regarding clustering and collaborative filtering.

Is it better to focus on the best-matched cluster to find neighbors, or to scan all of the cluster-centers? ClustKnn computes recommendations for the target user by seeking the closest neighbors from the cluster centers. However, another possibility is to first find the best-matched cluster for the target user, and then search for the best neighbors within the selected cluster only. We now provide three reasons to avoid this approach. First, this approach might hurt the *coverage* of the recommender, i.e., there can be more items with fewer personalized recommendations. The reason is that the users in the chosen cluster may not have rated a large fraction of the items that people in other clusters rated. Second, this approach might incur high computational cost similar to the regular user-based KNN, since the selected cluster can be a very large one. Third, as figure 5 shows, a large fraction of the closest neighbors may reside in other clusters than the one the target user belongs to. As a result, using a single cluster can easily lead to using less similar neighbors and thereby reducing accuracy. Note also from the figure that this problem gets worse as the number of clusters grows.

7 Conclusion

In this paper we have explored clustering to address scalability, a fundamental challenge to collaborative filtering recommender algorithms. In particular we have studied ClustKnn, a hybrid *memory-* and *model*-based collaborative filtering algorithm that is simple, intuitive, and highly scalable. The method achieves recommendation quality comparable to that of several other well-known CF algorithms. Further, the operator of the recommender system can tune a parameter in the model to trade off speed and scalability.

In the future, we plan to extend this approach to mitigate the so called *cold-start* problem [26] in CF. That is, a collaborative filtering recommender cannot produce personalized recommendations on newly introduced items lacking any or sufficient user-opinions on those items. By clustering on the space of item feature information, we hope to investigate the implications of building a *hybrid* recommender that works as a CF-based recommender on items with enough preference information, and as a content-based recommender otherwise.

8 Acknowledgments

We appreciate many helpful comments provided by Sheng Zhang of Dartmouth College in properly implementing SVD-based CF. Dan Cosley of GroupLens research was very helpful in giving feedback on early drafts of this paper. Shilad Sen's pLSA code was of great help. This work was supported by grants from the NSF(IIS 03-24851 and IIS 96-13960).

References

1. C. Basu, H. Hirsh, and W. Cohen. Recommendation as classification: using social and content-based information in recommendation. In *Proceedings of the 1998 National Conference on Artificial Intelligence (AAAI-98)*, pages 714–720, 1998.
2. J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 43–52, July 1998.
3. B. Cestnik. Estimating probabilities: A crucial task in machine learning. In *Proc. Ninth European Conference on Artificial Intelligence*, pages 147–149, 1990.
4. S. H. S. Chee, J. Han, and K. Wang. RecTree: An efficient collaborative filtering method. *Lecture Notes in Computer Science*, 2114, 2001.
5. C. Cleverdon, J. Mills, and M. Keen. *Factors Determining the Performance of Indexing Systems: ASLIB Cranfield Research Project. Volume 1: Design.* ASLIB Cranfield Research Project, Cranfield, 1966.
6. S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.
7. K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Inf.Retr.*, 4(2):133–151, 2001. ID: 187.
8. J. Herlocker, J. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 1999 Conference on Research and Development in Information Retrieval (SIGIR-99)*, Aug. 1999.
9. J. Herlocker, J. Konstan, L. Terveen, and J. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1):5–53, Jan. 2004.
10. T. Hofmann. Probabilistic latent semantic analysis. In *Proc. of Uncertainty in Artificial Intelligence, UAI'99*, Stockholm, 1999.
11. T. Hofmann. Latent semantic models for collaborative filtering. *ACM Trans. Inf. Syst.*, 22(1):89–115, 2004.

12. A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Comput. Surv.*, 31(3):264–323, 1999.
13. J. Kelleher and D. Bridge. Rectree centroid: An accurate, scalable collaborative recommender. In P. Cunningham, T. Fernando, and C. Vogel, editors, *Procs. of the Fourteenth Irish Conference on Artificial Intelligence and Cognitive Science*, pages 89–94, 2003.
14. G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.
15. B. Marlin. Modeling user rating profiles for collaborative filtering. In *NIPS*, 2003. crossref: DBLP:conf/nips/2003.
16. P. Melville, R. J. Mooney, and R. Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *Eighteenth national conference on Artificial intelligence*, pages 187–192. American Association for Artificial Intelligence, 2002. ID: 179.
17. B. Miller, I. Albert, S. K. Lam, J. A. Konstan, and J. Riedl. Movielens unplugged: Experiences with a recommender system on four mobile devices. In *Proceedings of the 17th Annual Human-Computer Interaction Conference (HCI 2003), British HCI Group*, Miami, FL, Sept. 2003.
18. O. Nasraoui and M. Pavuluri. Complete this puzzle: A connectionist approach to accurate web recommendations based on a committee of predictors. In *WebKDD-2004 workshop on Web Mining and Web Usage Analysis*, Seattle, WA, 2004.
19. D. M. Pennock, E. Horvitz, S. Lawrence, and C. L. Giles. Collaborative filtering by personality diagnosis: A hybrid memory and model-based approach. In *UAI '00: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 473–480, Stanford, CA, 2000. Morgan Kaufmann Publishers Inc.
20. 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 1994 ACM Conference on Computer Supported Cooperative Work*, pages 175–186, Chapel Hill, North Carolina, United States, 1994. ACM Press.
21. G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, Inc, 1986.
22. B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering. In *Fifth International Conference on Computer and Information Technology (ICCIT 2002).*
23. B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW '01: Proceedings of the 10th International Conference on World Wide Web*, pages 285–295, Hong Kong, 2001. ACM Press.
24. B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl. Analysis of recommender algorithms for e-commerce. In *ACM E-Commerce 2000*, pages 158 – 167, 2000.
25. B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl. Application of dimensionality reduction in recommender system – a case study. In *ACM WebKDD 2000 Web Mining for E-Commerce Workshop*, Boston, MA, USA, 2000.
26. A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 253–260, New York, NY, USA, 2002. ACM Press.
27. N. Srebro and T. Jaakkola. Weighted low rank approximation, 2003.
28. M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques, 2000.
29. K. Swearingen and S. Rashmi. Interaction design for recommender systems. In *Designing Interactive Systems 2002.* ACM, 2002.

30. L. Ungar and D. Foster. Clustering methods for collaborative filtering. In *Proceedings of the Workshop on Recommendation Systems. AAAI Press, Menlo Park California.*, 1998.
31. 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 *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 114–121, New York, NY, USA, 2005. ACM Press.
32. K. Yu, X. Xu, J. Tao, M. Ester, and H.-P. Kriegel. Instance selection techniques for memory-based collaborative filtering. In *SDM*, 2002.