

Efficient User Preference Predictions using Collaborative Filtering

Yang Song

Computer Science and Engineering,
Pennsylvania State University,
University Park, PA 16802, USA
yasong@cse.psu.edu

C. Lee Giles

Information Sciences and Technology,
Pennsylvania State University,
University Park, PA 16802, USA
giles@ist.psu.edu

Abstract

Two major challenges in collaborative filtering are the efficiency of the algorithms and the quality of the recommendations. A variety of machine learning methods have been applied to address these two issues, including feature selection, instance selection, and clustering. Most existing methods either compromise computational complexity or prediction precision. Two novel, scalable memory-based CF algorithms are proposed, namely BS1, BS2, which combine the strengths of existing techniques while discarding their weaknesses. Experiments show that both the efficiency and performance have been improved when compared to three classical techniques: VSIM, FCBF and PD.

1 Introduction

Web search has become a serious venture for many with consumers making economic decisions based on search results. As a result, predicting users preference and then giving useful recommendations with limited information gained from the users or their behavior has become a challenge for online stores and information providers. Collaborative filtering (CF) became a very popular technique not only for recommender systems in recent years, but also for research areas such as artificial intelligence, human computer interface, etc.

There are two major approaches of collaborative filtering, memory-based and model-based. Memory-based algorithms store users ratings/preferences in storage, and give recommendations based on known scores from existing users [1]. Model-based algorithms describe users preference by applying descriptive models to users and/or ratings; the virtue being that once the model is established, little computation is required for

prediction [2]. However, the training requirements of both models require trade-offs between on-line and off-line computation.

CF has the following issues - efficiency of the algorithms and quality of the recommendations. In general, the computational complexity of memory-based and model-based algorithms are $O(nm^2)$ and $O(nm)$, respectively, where m denotes the number of users and n denotes the number of items (e.g., movies) in the data sets where both have an upper bound of $m \times n$. Currently, data sets from large on-line stores contain millions of items as well as millions of registered users, which can be computationally expensive.

The quality of the recommendation is problematic and is primarily due to the number of items that users rated. Given the large number of items in the data sets, a single user may want to rate a very small portion of the items, resulting in most items unrated. For example, in EachMovie database, a user rates 20 movies on average, while the whole database contains more than 1,000 movies. Thus making recommendations from very limited data can lead to unacceptable errors.

In this paper, we propose two innovative algorithms for memory-based CF. The first algorithm, we denote as BS1, applies the instance selection technique to select the *best* candidates for prediction, based on an improved weight function. The second algorithm, denoted as BS2, unifies instance selection and feature selection techniques, and recommendations are implemented by clustering users into several featured clusters based on user profiles. We employ KMeans for our clustering algorithm and dynamically choose the best \mathcal{K} that leads to the optimal clustering results. The running time of BS2 equals that of BS1 in the log of the numbers of users.

2 Best Selection Algorithm (BS1)

For the related work of CF, we refer interested readers to the classic paper [1]. In this section, we give a formal definition of our memory-based CF algorithm.

2.1 A Revised Weight Function

Given a vector space V and a field of scalars K , an inner product is defined as a function $\langle \cdot, \cdot \rangle : V \times V \rightarrow K$ which satisfies the properties of *linearity*, *conjugate symmetry* and *positive definite*. Generally, an inner product is a generalization of the dot product. Usually in a vector space, inner product is used as a way to multiply vectors together, with the result of this multiplication being a scalar. In the area of information retrieval, inner product is used as a measurement of vector similarity that represents how similar two or more queries/sentences are to each other. Based on these features, we introduce a new weight function as follows:

$$w(a, i) = \frac{\sum_{j=0}^m \langle \min(V_{aj}, \epsilon_a), \min(V_{ij}, \epsilon_i) \rangle, Sim(a, i, j))}{\min(|I_a|, |I_j|)} \quad (1)$$

Where function Sim is defined as:

$$Sim(a, i, j) = \frac{(S - |V_{aj} - V_{ij}|)}{S} \quad (2)$$

Here in equation (1) and equation (2), a is the active user we want to predict, V_{ij} represents the rating of user i over item j , I_i denotes the number of items that user i rated, m equals to the total number of items and S stands for the rating scale. The vector $\epsilon = \{\epsilon_1, \epsilon_2, \dots, \epsilon_m\}$ are some significant small coefficients that are a little bit smaller than the lowest feature frequencies for each sample. Specifically, $\epsilon(\mathcal{F}) = \min_{r \neq 0} \frac{\|\mathcal{F}_r\|_1}{\|r\|_1}$. The reason that instead of using the true value of V_{ij} , we use $\min(V_{ij}, \epsilon_i)$ in equation (2) for computing the weight function is that we are only concerned about whether users a and i have rated item j or not. How close their ratings are related is decided by the function $Sim(\cdot)$, which computes the similarity between two ratings by calculating the difference between user ratings divided by the rating scale resulting in vector distances (or similarity). It is evident that the higher the similarity, the closer relationships users have to each other. Obviously $Sim(a, a, i) = 1$ for all users, indicating that the similarity between a user and itself is the highest.

2.2 Algorithm and Analysis

In algorithm Best Selection (BS1), m and n are used to denote the number of users and the number of items,

respectively. The idea behind this algorithm is that instead of doing computation across the entire datasets only for a single prediction, we choose totally $\lceil \log m \rceil$ users that has the *best* similarity with the active user a for prediction. i.e., these candidates are the best ones to represent user a 's ratings of the items in the same data set. To achieve the best result, the algorithm is iterated $\lceil \log m \rceil$ times, at each time $\lceil \log m \rceil$ users are randomly picked, and only the *best* one that has the highest similarity (judged by $tw(a, i)$) with the active user a is selected and inserted into the candidates list.

Algorithm 1 Best Selection Algorithm (BS1)

input: user-item matrix $\mathcal{P} \in \mathcal{R}^{m \times n}$
weight matrix $\mathcal{W} \in \mathcal{R}^{m \times m}$
output: candidates list $\mathcal{C} \in \mathcal{R}^{\lceil \log m \rceil \times n}$
Initialize $\mathcal{C} \leftarrow \emptyset$
for each active user a **do**
 for i equals 0 to $\lceil \log m \rceil$ **do**
 randomly choose $\lceil \log m \rceil$ users \mathcal{U} from \mathcal{P}
 select the candidate μ that satisfies:
 $\forall v \in \mathcal{U}$ and $v \neq \mu, \mathcal{W}(a, \mu) > \mathcal{W}(a, v)$
 $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \mathcal{P}(\mu)$
 end
end
output \mathcal{C}

Once we have the candidates list and the weight function, we can predict user a 's rating for item j as follows (remember \mathcal{C} is the candidates list):

$$\mathcal{Q}(a, j) = \sum_{c=0}^{\lceil \log m \rceil} \mathcal{W}(a, c) * \mathcal{P}(c, j), c \in \mathcal{C} \quad (3)$$

Some previous memory-based approaches [1] predicted user ratings by collecting ratings from all users in the datasets. Two shortcomings are obvious here: first, lots of noise factors were introduced to the prediction process. i.e., the ratings of those users whom have very little similarities with the active user a are also added up to predict its ratings, as long as the weights between these users and a are greater than 0. Secondly, time are wasted to compute these useless ratings, which have very minor effects to the final prediction results.

By adopting instance selection, we minimize the noise factors by selecting the most representative users from the data set. As a result, on establishing the candidates list, we could get better prediction results by computing from a much smaller set, but definitely no less reliable, of related users, regardless of items. The running time is also significantly reduced to $O(n \log^2 m)$.

It has also been realized that by using random selection, we may get the optimal result for each step, but we may not achieve the optimum solution globally. i.e.,

those users that have the highest weights with the active user have chances of not being selected as candidates. However, trying to find the absolute best candidates each time would be labor-intensive and time-consuming. By mentioning *best* here, we mean that our approach finds the best trade-off between computational complexity and prediction results. i.e., by selecting the best candidate for each step and repeating the same step $\lceil \log m \rceil$ times, we have minimized the standard deviation of errors that could cause mis-prediction.

3 Clustering Approach — Improved Best Selection Algorithm (BS2)

Generally, feature selection is different from instance selection regarding selection criteria. i.e., instances are selected based on user similarities while features are chosen according to the quantity of ratings provided by users. In algorithm 2, we want to select those features that have been rated most frequently by users. At the beginning of BS2, a candidates list is retrieved by using BS1. Based on that, we count the number of items that have been rated by the users in the candidates list, and select the first $\lceil \log n \rceil$ features according to the descending order of the frequency.

Algorithm 2 Best Selection Algorithm (BS2)

input: user-item matrix $\mathcal{P} \in \mathcal{R}^{m \times n}$

output: instance-feature list $\mathcal{F} \in \mathcal{R}^{\lceil \log m \rceil \times \lceil \log n \rceil}$

Initialize $\mathcal{C} \leftarrow \text{BS1}(\mathcal{P})$, $\mathcal{F} \leftarrow \emptyset$, $\mathcal{I} \leftarrow \emptyset$

for i equals 0 to n **do**

for j equals 0 to $\lceil \log m \rceil$ **do**

if $\mathcal{C}(j, i) > 0$

$\mathcal{I}(i) \leftarrow \mathcal{I}(i) + 1$

end

end

Append active user a to \mathcal{C} : $\mathcal{C} \leftarrow \mathcal{P}(a)$

for i equals 0 to $\lceil \log n \rceil$ **do**

 select the feature κ from \mathcal{C} that satisfies:

$\forall \xi \in \mathcal{C}$ and $\xi \neq \kappa, \mathcal{I}(\kappa) > \mathcal{I}(\xi)$

$\mathcal{F}_i \leftarrow \mathcal{F}_i \cup \mathcal{C}(\kappa)$

$\mathcal{C} \leftarrow \mathcal{C} - \mathcal{C}_\kappa$

end

The initialization part of algorithm 2 takes $O(n \log^2 m)$ time with respect to the number of users m and number of features n . The first part of the algorithm computes the number of times that the items have been rated by the selected candidates, taking $O(n \log m)$ time to finish. The second part of the algorithm selects $\lceil \log n \rceil$ features from all the items, requiring no more than $O(\log n)$ time. Considering in normal cases, $n \gg m$ and $m \gg n$, the running time of algorithm 2 is still bounded by $O(n \log^2 m)$.

3.1 Clustering for Predicting User Ratings

After applied to BS2, the sample data collected have been optimized for prediction. The prediction is then carried out by clustering the training data set based on the user profiles. In practice, the size of the raw data sets varies a lot. As a result, how to dynamically choose the number of clusters that could lead the results of clustering to the best prediction outcome becomes the key part of employing the classical clustering algorithm.

Several approaches have been applied to cluster the data sets of user-item matrices, which were mentioned in [3] as repeated clustering. In our approach, users who have similar preferences of the same items are clustered together. To determine the quality of clustering results, two metrics namely the average intra-class compactness and inter-class looseness of clusters are employed:

$$\mathcal{C}_p(\mathcal{K}) = \frac{1}{\mathcal{K}} \sum_{i=1}^{\mathcal{K}} \sum_{j=1}^{N_i} (\mathcal{C}(i) - \mathcal{U}(i))^2 \quad (4)$$

$$\mathcal{L}_s(\mathcal{K}) = \frac{1}{\mathcal{K}} \sum_{i=1}^{\mathcal{K}} \sum_{j=1}^{\mathcal{K}} (\mathcal{C}(i) - \mathcal{C}(j))^2 \quad (5)$$

In eq. (4) and (5), $\mathcal{U}(i) \in \mathcal{F}$, where \mathcal{F} is the result from BS2 and $\mathcal{F} \in \mathcal{R}^{\lceil \log m \rceil \times \lceil \log n \rceil}$, \mathcal{K} denotes the number of clusters, N_i represents the total number of users in class i , and $\mathcal{C}(i)$ is the centroid of class i which is defined as: $\mathcal{C}(i) = \frac{1}{N_i} \sum_{j=1}^{N_i} \mathcal{U}(i)$.

The purpose of clustering is to minimize the average intra-class compactness while simultaneously maximizing inter-class looseness. In Algorithm 3, two initial thresholds ϵ_1, ϵ_2 are chosen for intra-class compactness and inter-class looseness, respectively. The number of clusters \mathcal{K} are initialized to be half the number of instances selected from BS1. Each time when the KMeans algorithm is called, we get the labels for instances ι , cluster centroids c and the modified instance-feature list \mathcal{F}' which appends the labels to the last column of \mathcal{F} . After that, new $\tilde{\rho}$ and $\tilde{\tau}$ are computed by using the new labels. If both conditions are satisfied, we take the clustering results and compute the prediction value of the active user a . However, if the computed $\tilde{\rho}$ is larger than the initial threshold, we recognize that the current clusters are not compact enough, i.e., users that are clustered in the same class may still exhibit different user profiles thus has to be further split.

In the third case, if the value of $\tilde{\rho}$ is small enough, but $\tilde{\tau}$ is smaller than the threshold (which means we have "over-cluster" the users and clusters are not maximally separated), then clusters are merged to get larger inter-class looseness values.

Algorithm 3 Clustering for Prediction

input: instance-feature list $\mathcal{F} \in \mathcal{R}^{\lceil \log m \rceil \times \lceil \log n \rceil}$
output: rating-prediction list $\mathcal{P}' \in \mathcal{R}^{\lceil \log m \rceil}$
Initialize $\mathcal{K} \leftarrow \lceil \log m / 2 \rceil, \rho \leftarrow \epsilon_1, \tau \leftarrow \epsilon_2,$
 $\iota \leftarrow \emptyset, c \leftarrow \emptyset, \mathcal{P}' \leftarrow \emptyset$
while $\mathcal{K} > 2$ and $\mathcal{K} < \lceil \log m \rceil - 1$ **do**
 $(\iota, c, \mathcal{F}') \leftarrow \text{KMeans}(\mathcal{F}, \mathcal{K})$
 for i equals 1 to N_i **do**
 for j equals 1 to N_i **do**
 $\tilde{\rho} \leftarrow \mathcal{C}_p(\mathcal{F}')$
 $\tilde{\tau} \leftarrow \mathcal{L}_s(\mathcal{F}')$
 end
 end
 if $\tilde{\rho} < \rho$ and $\tilde{\tau} > \tau$ **break**
 else if $\tilde{\rho} > \rho$
 $\mathcal{K} \leftarrow \mathcal{K} + 1$
 else if $\tilde{\rho} < \rho$ and $\tilde{\tau} < \tau$
 $\mathcal{K} \leftarrow \mathcal{K} - 1$
 end while
for i equals 1 to $\lceil \log m \rceil$ **do**
 for j equals 1 to n **do**
 if $\iota_i = \iota_{\lceil \log m \rceil + 1}$
 $\mathcal{P}'_j \leftarrow \mathcal{P}'_j + \mathcal{W}(i, \lceil \log m \rceil + 1) \mathcal{P}(\lceil \log m \rceil + 1, j)$
 end
 end
end

4 Experiments and Discussion

Performance of our algorithms are evaluated based on computational complexity and prediction accuracy, two benchmark datasets from GroupLens are employed. Comparisons are made to [1], Fast Correlation-Based Filter [4] (FCBF) and Personality Diagnosis [2] (PD).

Three protocols are employed for evaluation *all but one*, *given two* and *give ten*. For evaluation metrics, we employ the *MAE* (Mean Absolute Error) score.

4.1 Results and Discussions

Table 1 exhibits the experiment results. BS1 and BS2 perform better than PD, FCBF and VSIM in all three protocols for all test data sets. The average improvements of BS1 and BS2 regarding MSE, RS and MUG scores are 12.7%, 20.4% and 13.5%, by comparing to the average scores of the three algorithms.

Table 2 shows the running time of these algorithms. BS1 and BS2 require significantly less time than the other three algorithms, especially for the large data sets.

Table 3 summarizes the performance comparison between BS1 and BS2. We use the measurement of how

Datasets	Algorithms	Protocols		
		<i>AllBut1</i>	<i>Given10</i>	<i>Given2</i>
MovieLen1	PD	0.964	0.986	1.039
	FCBF	0.999	1.069	1.296
	VSIM	2.136	2.235	2.113
	BS1	0.825	0.826	0.878
	BS2	0.814	0.835	1.022
MovieLen2	PD	1.023	1.011	1.125
	FCBF	1.001	1.068	1.265
	VSIM	2.345	2.274	2.256
	BS1	1.078	1.079	1.079
	BS2	0.802	0.811	0.978

Table 1. MSE Scores

Datasets	Running Time (in ms)				
	VSIM	PD	FCBF	BS1	BS2
MovieLen1	4833	3200	4587	2544	3517
MovieLen2	15669	5644	10238	4233	5291

Table 2. Algorithm Running Time.

much can BS2 improve from BS1. We notice that BS2 performs better on MSE but cost more computational power. In BS2, we call KMeans several times with different value of cluster numbers \mathcal{K} . Since BS2 uses the results (candidate lists) from BS1, it is then reasonable for BS2 to cost a little bit more time than BS1.

References

- [1] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering.
- [2] 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*.
- [3] L. Ungar and D. Foster. Clustering methods for collaborative filtering. In *Proceedings of the Workshop on Recommendation Systems, 1998*.
- [4] L. Yu and H. Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *ICML '03*.

	Synthetic	MovieLen1	MovieLen2	Overall
MSE	5.6%	4.7%	9.2%	6.5%
Time	-3.2%	-29%	-19.9%	-17.4%

Table 3. Improvement of BS2 over BS1.