Nearest-Biclusters Collaborative Filtering with Constant Values

Panagiotis Symeonidis, Alexandros Nanopoulos, Apostolos Papadopoulos, and Yannis Manolopoulos

Aristotle University, Department of Informatics, Thessaloniki 54124, Greece
{symeon,alex,apostol,manolopo}@delab.csd.auth.gr

Abstract. Collaborative Filtering (CF) Systems have been studied extensively for more than a decade to confront the "information overload" problem. Nearest-neighbor CF is based either on common user or item similarities, to form the user's neighborhood. The effectiveness of the aforementioned approaches would be augmented, if we could combine them. In this paper, we use biclustering to disclose this duality between users and items, by grouping them in both dimensions simultaneously. We propose a novel nearest-biclusters algorithm, which uses a new similarity measure that achieves partial matching of users' preferences. We apply nearest-biclusters in combination with a biclustering algorithm – Bimax – for constant values. Extensively performance evaluations on two real data sets is provided, which show that the proposed method improves the performance of the CF process substantially. We attain more than 30% and 10% improvement in terms of precision and recall, respectively.

1 Introduction

Information Filtering has become a necessary technology to attack the "information overload" problem. In our everyday experience, while searching on a topic (e.g., products, movies, etc.), we often rely on suggestions from others, more experienced in it. In the Web, however, the plethora of available suggestions renders it difficult to detect the trustworthy ones. The solution is to shift from individual to collective suggestions. Collaborative Filtering (CF) applies information retrieval and data mining techniques to provide recommendations based on suggestions of users with similar preferences. CF is a very popular method in recommender systems and e-commerce applications.

1.1 Motivation

Two families of CF algorithms have been proposed in the literature: (a) nearest-neighbors (a.k.a. memory-based) algorithms, which recommend according to the preferences of nearest neighbors; and (b) model-based algorithms, which recommend by first developing a model of user ratings. Related research has reported

O. Nasraoui et al. (Eds.): WebKDD 2006, LNAI 4811, pp. 36–55, 2007.

that nearest-neighbor algorithms present good performance in terms of accuracy. Nevertheless, their main drawback is that they cannot handle scalability to large volumes of data. On the other hand, model-based algorithms, once they have build the model, present good scalability. However, they have the overhead to build and update the model, and they cannot cover as diverse a user range as the nearest-neighbor algorithms do [29]. Therefore, a first goal is to develop nearest-neighbor algorithms that combine good accuracy with the advantage of scalability that model-based algorithms present.

Regarding nearest-neighbor algorithms, there exist two main approaches: (a) *user-based* (UB) CF, which forms neighborhoods based on similarity between users; and (b) *item-based* (IB) CF, which forms neighborhoods based on similarities between items. However, both UB and IB are one-sided approaches, in the sense that they examine similarities either only between users or only between items, respectively. This way, they ignore the clear duality that exists between users and items. Furthermore, UB and IB algorithms cannot detect partial matching of preferences, because their similarity measures consider the entire set of items or users, respectively. However, two users may share similar preferences only for a subset of items. For instance, consider two users that share similar preferences for science-fiction books and differentiate in all other kinds of literature. In this case, their partial matching for science-fiction, which can help to provide useful recommendations between them for this kind of books, will be missed by existing approaches. Therefore, by measuring similarity with respect to the entire set of items, we miss partial matchings between two users, since the differences in the remaining items prevails over the subset of items in which their preferences match. Analogous reasoning applies for the IB case. Thus, a second goal is to develop nearest-neighbor algorithms that will be able to consider the duality between users and items, and at the same time, to capture partial matching of preferences.

Finally, the fact that a user usually has various different preferences, has to be taken into account for the process of assigning him to clusters. Therefore, such a user has to be included in more than one clusters. Notice that this cannot be achieved by most of the traditional clustering algorithms, which place each item/user in exactly one cluster. In conclusion, a third goal is to adopt an approach that does not follow the aforementioned restriction and can cover the entire range of the user's preferences.

1.2 Contribution

To attain the first described goal, i.e., to develop scalable nearest-neighbor algorithms, we propose the grouping of different users or items into a number of clusters, based on their rating patterns. This way, similar searching is performed efficiently, because we use consolidated information (that is, the clusters) and not individual users or items.

To address the second described goal, i.e., to disclose the duality between users and items, we propose the generation of groups of users *and* items at the same time. The simultaneous clustering of users and items discovers *biclusters*, which correspond to groups of users which exhibit highly correlated ratings on groups of items. Biclusters allow the computation of similarity between a test user and a bicluster *only* on the items that are included in the bicluster. Thus, partial matching of preferences is taken into account too. Moreover, a user can be matched with several nearest biclusters, thus to receive recommendations that cover the range of his various preferences.

To face the third described goal, i.e., to include a user in more than one clusters, we allow a degree of overlap between biclusters. Thus, if a user presents different item preferences, by using overlapping biclusters, he can be included in more clusters in order to cover all his different preferences.

The contributions of this paper are summarized as follows:

- To disclose the duality between users and items and to capture the range of the user's preferences, we introduce for the first time, to our knowledge, the application of an exact biclustering algorithm to the CF area.
- We propose a novel nearest-biclusters algorithm, which uses a new similarity measure that achieves partial matching of users' preferences.
- Our extensive experimental results illustrate the effectiveness and efficiency of the proposed algorithm over existing approaches.

The rest of this paper is organized as follows. Section 2 summarizes the related work, whereas Section 3 contains the analysis of the CF issues. The proposed approach is described in Section 4. Experimental results are given in Section 5. Finally, Section 6 concludes this paper.

2 Related Work

In 1992, the Tapestry system [6] introduced Collaborative Filtering (CF). In 1994, the GroupLens system [21] implemented a CF algorithm based on common users preferences. Nowadays, it is known as user-based CF algorithm, because it employs users' similarities for the formation of the neighborhood of nearest users. Since then, many improvements of user-based algorithm have been suggested, e.g., [8,18,23].

In 2001, another CF algorithm was proposed. It is based on the items' similarities for a neighborhood generation [24,13,3]. Now, it is denoted as item-based or item-item CF algorithm, because it employs items' similarities for the formation of the neighborhood of nearest users.

The concept of biclustering has been used in [17] to perform grouping in a matrix by using both rows and columns. However, biclustering has been used

previously in [7] under the name *direct clustering*. Recently, biclustering (also known as *co-clustering*, *two-sided clustering*, *two-way clustering*) has been exploited by many researchers in diverse scientific fields, towards the discovery of useful knowledge [2,4,5,14,19]. One of these fields is bioinformatics, and more specifically, microarray data analysis. The results of each microarray experiment are represented as a data matrix, with different samples as rows and different genes as columns. Among the proposed biclustering algorithms we highlight the following: (i) Cheng and Churchs algorithm [2] which is based on a mean squared residue score, (ii) the Iterative Signature Algorithm (ISA) which searches for submatrices representing fix points [12], (iii) the Order-Preserving Submatrix Algorithm (OPSM), which tries to identify large submatrices for which the induced linear order of the columns is identical for all rows [1],(iv) the Samba Algorithm, which is a graph theoretic approach in combination with a statistical model [27,26], and (v) the Bimax algorithm, an exact biclustering algorithm based on a divide-and-conquer strategy, that is capable of finding all maximal bicliques in a corresponding graph-based matrix representation [20].

In the CF area, there is no related work that has applied a specific biclustering algorithm to provide recommendations. Madeira and Oliveira [15] have reported in their survey, the existence of works that have used two-sided clustering in the CF field. In these models [28,11], there is a hidden variable for each user and item, respectively, that represents the cluster of that user or item. For each user-item pair, there is a variable that denotes their relation. The existence of the relation depends on the cluster of the person, and the cluster of item, hence the notion of two-sided clustering. These are latent class models using statistical estimation of the model parameters and clustering is performed separately for users and items. In contrast, our approach is based on the application of specific biclustering algorithms[1] that perform simultaneous clustering of users and items.

3 Examined Issues

In this section, we provide details for the issues we examine about CF algorithms. Table 1 summarizes the symbols that are used in the sequel.

Scalability: Scalability is important, because in real-world applications the number of users/items is very large. As the number of users/items grows, CF algorithms face performance problems. Therefore, CF algorithms should be evaluated in terms of their responding time in providing recommendations.

Similarity measure: The most extensively used similarity measures are based on correlation and cosine-similarity [9,24]. Specifically, user-based CF algorithms mainly use Pearson's Correlation (Equation 1), whereas for item-based

[1] For implementation issues, we use the Bimax biclustering algorithm, however any other algorithm can be used equally well, as our approach is independent of the specific biclustering algorithm that is used.

Table 1. Symbols and definitions

Symbol	Definition
k	number of nearest neighbors or biclusters
N	size of recommendation list
P_τ	threshold for positive ratings
$\mathcal{I}$	domain of all items
$\mathcal{U}$	domain of all users
u, v	some users
i, j	some items
I_u	set of items rated by user u
U_i	set of users rated item i
$r_{u,i}$	the rating of user u on item i
$\overline{r}_u$	mean rating value for user u
$\overline{r}_i$	mean rating value for item i
n	minimum allowed number of users in a bicluster
m	minimum allowed number of items in a bicluster
$\mathcal{B}$	set of all biclusters
b	a bicluster
I_b	set of items of bicluster b
U_b	set of users of bicluster b

CF algorithms, the Adjusted Cosine Measure is preferred (Equation 2) [16,24]. The Adjusted Cosine Measure is a variation of the simple cosine formula, that normalizes bias from subjective ratings of different users. As default options, for user-based CF we use the Pearson Correlation, whereas for item-based we use the Adjusted Cosine Similarity, because they presented the best behavior overall.

$$\text{sim}(u,v) = \frac{\sum_{\forall i \in S} (r_{u,i} - \overline{r}_u)(r_{v,i} - \overline{r}_v)}{\sqrt{\sum_{\forall i \in S} (r_{u,i} - \overline{r}_u)^2} \sqrt{\sum_{\forall i \in S} (r_{v,i} - \overline{r}_v)^2}}, S = I_u \cap I_v. \quad (1)$$

$$\text{sim}(i,j) = \frac{\sum_{\forall u \in T} (r_{u,i} - \overline{r}_u)(r_{u,j} - \overline{r}_u)}{\sqrt{\sum_{\forall u \in U_i} (r_{u,i} - \overline{r}_u)^2} \sqrt{\sum_{\forall u \in U_j} (r_{u,j} - \overline{r}_u)^2}}, T = U_i \cap U_j. \quad (2)$$

Neighborhood size: The number, k, of nearest neighbors used for the neighborhood formation is important, because it can affect substantially the system's accuracy. In most related works [8,22], k has been examined in the range of values between 10 and 100. The optimum k depends on the data characteristics (e.g., sparsity). Therefore, CF algorithms should be evaluated against varying k, in order to tune it.

Positive rating threshold: Recommendation for a test user is performed by generating the top-N list of items that appear most frequently in his formed neighborhood (this method is denoted as Most-Frequent item-recommendation). Nevertheless, it is evident that recommendations should be "positive", as it is not success to recommend an item that will be rated with, e.g., 1 in 1-5 scale [25]. Thus, "negatively" rated items should not contribute to the increase of accuracy. We use a rating-threshold, P_τ, to recommended items whose rating is not less than this value. If we do not use a P_τ value, then the results become misleading.

Training/Test data size: There is a clear dependence between the training set's size and the accuracy of CF algorithms [24]. Through our experimental study we verified this conclusion. Though most related research uses a size around 80%, there exist works that use significantly smaller sizes [16]. Therefore, CF algorithms should be evaluated against varying training data sizes.

Recommendation list's size: The size, N, of the recommendation list corresponds to a tradeoff: With increasing N, the absolute number of relevant items (i.e., recall) is expected to increase, but their ratio to the total size of the recommendation list (i.e., precision) is expected to decrease. (Recall and precision metrics are detailed in the following.) In related work [13,24], N usually takes values between 10 and 50.

Evaluation Metrics: Several metrics have been used for the evaluation of CF algorithms, for instance the Mean Absolute Error (MAE) or the Receiving Operating Characteristic (ROC) curve [9,10]. MAE represents the absolute differences between the real and the predicted values and is an extensively used metric. From our experimental study (Section 5) we understood that MAE is able to characterize the accuracy of prediction, but is not indicative for the accuracy of recommendation. Since in real-world recommender systems the experience of users mainly depends on the accuracy of recommendation, MAE may not be the preferred measure. For this reason we focus on widely accepted metrics from information retrieval.

For a test user that receives a top-N recommendation list, let R denote the number of *relevant recommended items* (the items of the top-N list that are rated higher than P_τ by the test user). We define the following:

- *Precision* is the ratio of R to N.
- *Recall* is the ratio of R to the total number of relevant items for the test user (all items rated higher than P_τ by him).

Notice that with the previous definitions, when an item in the top-N list is not rated at all by the test user, we consider it as *irrelevant* and it counts negatively to precision (as we divide by N) [16]. In the following we also use F_1, because it combines both the previous metrics:

$$\mathrm{F}_1 = 2 \cdot \mathrm{recall} \cdot \mathrm{precision} / (\mathrm{recall} + \mathrm{precision}).$$

4 Nearest Bicluster Approach

4.1 Outline of the Proposed Approach

Our approach consists of three stages.

- *Stage 1*: the data preprocessing/discretization step.
- *Stage 2*: the biclustering process.
- *Stage 3*: the nearest-biclusters algorithm.

The proposed approach, initially, applies a data preprocessing/discretization step. The motivation is to preserve only the positive ratings. Consequently, we proceed to the biclustering process, where we create simultaneously groups consisting of users and items. Finally, we implement the k nearest-biclusters algorithm. We calculate similarity between each test user and the generated bicluster. Thus, we create the test users' neighborhood, consisted of the k nearest biclusters. Then, we provide for each test user a Top-N recommendation list based on the most frequent items in his neighborhood.

To ease the discussion, we will use the running example illustrated in Figure 1, where I_{1-7} are items and U_{1-9} are users. As shown, the example data set is divided into training and test set. The null cells (no rating) are presented with dash.

	I_1	I_2	I_3	I_4	I_5	I_6	I_7
U_1	5	-	2	-	1	-	-
U_2	2	-	4	1	4	3	-
U_3	4	-	2	-	2	-	5
U_4	-	3	1	4	-	5	2
U_5	-	2	4	2	5	1	-
U_6	5	1	-	1	-	-	3
U_7	-	2	5	-	4	1	-
U_8	1	4	-	5	4	3	-

(a)

	I_1	I_2	I_3	I_4	I_5	I_6	I_7
U_9	5	-	4	-	1	-	2

(b)

Fig. 1. Running example: (a) training Set; (b) test Set

4.2 The Data Preprocessing/Discretization Step

Data preprocessing is applied to make data more suitable for data mining. According to the positive rating threshold, we have introduced in Section 3, recommendations should be "positive", as it is not success to recommend an item that

	I_1	I_2	I_3	I_4	I_5	I_6	I_7
U_1	5	-	-	-	-	-	-
U_2	-	-	4	-	4	3	-
U_3	4	-	-	-	-	-	5
U_4	-	3	-	4	-	5	-
U_5	-	-	4	-	5	-	-
U_6	5	-	-	-	-	-	3
U_7	-	-	5	-	4	-	-
U_8	-	4	-	5	4	3	-

Fig. 2. Training Set with rating values $\geq P_\tau$

	I_1	I_2	I_3	I_4	I_5	I_6	I_7
U_1	1	0	0	0	0	0	0
U_2	0	0	1	0	1	1	0
U_3	1	0	0	0	0	0	1
U_4	0	1	0	1	0	1	0
U_5	0	0	1	0	1	0	0
U_6	1	0	0	0	0	0	1
U_7	0	0	1	0	1	0	0
U_8	0	1	0	1	1	1	0

Fig. 3. Binary discretization of the Training Set

will be rated with, e.g., 1 in 1-5 scale. Thus, "negatively" rated items should not contribute to the increase of accuracy. This is the reason that we are interested only in the positive ratings, as shown in Figure 2.

Furthermore, as biclustering groups items and users simultaneously, it allows to identify sets of users sharing common preferences across subsets of items. In our approach, the main goal is to find the largest possible subsets of users that have rated positively (above P_τ rating threshold) items. Therefore, the problem can be discretized to binary values by setting as discretization threshold the P_τ rating threshold. The binarized data are shown in Figure 3.

Notice that binarization of data is optional and can be omitted, in case we use a biclustering algorithm which discovers biclusters with coherent values on both users and items. In our case, as shown in the next subsection, we use Bimax algorithm which finds clusters with constant values and the binarization step is required. In a future work, we will examine more types of biclustering algorithms, which will omit the preprocessing step.

4.3 The Biclustering Process

The biclustering process on a data matrix involves the determination of a set of clusters taking into account both rows and columns. Each bicluster is defined on

	I_4	I_2	I_6	I_5	I_3	I_1	I_7
U_3	0	0	0	0	0	1	1
U_6	0	0	0	0	0	1	1
U_5	0	0	0	1	1	0	0
U_7	0	0	0	1	1	0	0
U_2	0	0	1	1	1	0	0
U_8	1	1	1	1	0	0	0
U_4	1	1	1	0	0	0	0
U_1	0	0	0	0	0	1	0

Fig. 4. Applying biclustering to the Training Set

a subset of rows and a subset of columns. Moreover, two biclusters may overlap, which means that several rows or columns of the matrix may participate in multiple biclusters. Another important characteristic of biclusters is that each bicluster should be maximal, i.e., it should not be fully contained in another determined bicluster.

For the biclustering step, there are two main bicluster classes that have been proposed: (a) biclusters with constant values and (b) biclusters with coherent values. The first category looks for subsets of rows and subsets of columns with constant values, while the second is interested in biclusters with coherent values. For the biclustering step, we have adopted a simple constant biclustering algorithm denoted as Bimax [20], which is executed off-line. It is an exact biclustering algorithm based on a divide-and-conquer strategy that is capable of finding all maximal biclusters in a corresponding graph-based matrix representation.

For the Bimax algorithm, a bicluster $b(U_b, I_b)$ corresponds to a subset of users $U_b \subseteq \mathcal{U}$ that jointly present positively rating behavior across a subset of items $I_b \subseteq \mathcal{I}$. In other words, the pair (U_b, I_b) defines a submatrix for which all elements equal to 1.

The main goal of the Bimax algorithm is to find all biclusters that are *inclusion-maximal*, i.e, that are not entirely contained in any other bicluster. The required input to Bimax is the minimum number of users and the minimum number of items per bicluster. It is obvious that the Bimax algorithm finds a large number of overlapping biclusters. To avoid this we can perform a secondary filtering procedure to reduce this number to the desired overlapping degree.

In Figure 4, we have applied the Bimax algorithm to the running example. Four biclusters are found (depicted with dashed rectangles), with minimum number of users equal to 2 (i.e., $|U_b| \geq 2$) and the minimum number of items equal to 2 (i.e., $|I_b| \geq 2$). These bilcusters are summarized as follows:

$$
\begin{aligned}
b_1\colon U_{b_1} &= \{U_3, U_6\}, & I_{b_1} &= \{I_1, I_7\} \\
b_2\colon U_{b_2} &= \{U_5, U_7, U_2\}, & I_{b_2} &= \{I_5, I_3\} \\
b_3\colon U_{b_3} &= \{U_2, U_8\}, & I_{b_3} &= \{I_6, I_5\} \\
b_4\colon U_{b_4} &= \{U_8, U_4\}, & I_{b_4} &= \{I_4, I_2, I_6\}
\end{aligned}
$$

We have to notice that there is overlap between biclusters. Specifically, between biclusters 2 and 3 in item I_5. Also, we have overlapping between biclusters 3 and 4 in item I_6. We can allow this overlapping (it reaches 16,6%) or we can forbid it. If we forbid it, then we will abolish the existence of the third bicluster because it is smaller than the other two. In order not to miss important biclusters, we allow overlapping. However, overlapping introduces a trade-off: (a) with few biclusters the effectiveness reduces, as several biclusters may be missed; (b) with a high number of biclusters efficiency reduces; as we have to examine many possible matchings. In our experimental results we show the tuning of the allowed overlapping factor.

4.4 The Nearest Bicluster Algorithm

In order to provide recommendations, we have to find the biclusters containing users with preferences that have strong partial similarity with the test user. This stage is executed on-line and consists of two basic operations:

- The formation of the test user's neighborhood, i.e., to find the k nearest biclusters.
- The generation of the top-N recommendation list.

```
Array k-NearestBiclusters(nB, IB[nB][nI], UI[nI])
begin
//int nI number of items
//int nB number of biclusters
//int cI, ncI common items/items not in common
//Array IB[nB][nI] stores items per bicluster (binary)
//Array UI[nI] stores the user ratings
//Array SIM[nB] stores user-biclusters similarities
for b=1 to nB
    cI=0; ncI=0; SIM[b] = 0;

    for i=1 to nI
       if (IB[b][i] = 1) and (UI[i] ≥ Pτ)
           cI = cI + 1;
       if (IB[b][i] = 1) and (UI[i] < Pτ)
           ncI=ncI+1;
       SIM[b] = cI/ (cI + ncI);

sort(SIM); //descending order
return (SIM[0..k-1]);
end
```

Fig. 5. The algorithm for the formation of a test user's biclusters neighborhood

To find the k nearest biclusters, we measure the similarity of the test user and each of the biclusters. The central difference with the past work is that we are interested in the similarity of test user and a bicluster *only* on the items that are included in the bicluster and not on all items that he has rated. As described, this allows for the detection of partial similarities. The similarity between the test user and each bicluster is calculated by dividing the items they have in common to the sum of items they have in common and not in common. In Equation 3, we calculate the similarity between a user u and bicluster b as follows:

$$sim(u, b) = \frac{|I_u \bigcap I_b|}{|I_u \bigcap I_b| + |I_b - I_u|} \tag{3}$$

It is obvious that similarity values range between [0,1]. The algorithm for the formation of the similarity matrix between a test user and the biclusters is shown in Figure 5.

```
Array TOPN(nI, nnB, topN, UB[nB], SIM[k])
begin
//int nI number of test users/items
//int topN number of items in recommendation list
//int nnB number of nearest biclusters
//Array IB[nB][nI] stores items per bicluster(binary)
//Array WF[nI] stores items' Weighted Frequency
//Array SIM[nB] stores users-biclusters similarities
//Array UB[nB] stores the number of users per bicluster
//Array TOPN[topN] stores the recommendation list
    for j=1 to nI
        WF[j].value = 0;
        WF[j].position = j;

    for b=1 to nnB

        for j=1 to nI
            //if an item belongs to the bicluster
            if (IB[SIM[b].position][j] > 0)
                WF[j].value += UB[b] * SIM[b];

    sort(WF); //descending order of WF values

    for i=1 to topN
        if (WF[i].value > 0)
            TOPN[i]= WF[i].position;
end
```

Fig. 6. Generation of top-N recommendation list

In the next phase, we proceed to the generation of the top-N recommendation list. For this purpose, we have to find the appearance frequency of each item and recommend the N most frequent. In Equation 4, we define as *Weighted Frequency* (WF) of an item i in a bicluster b, the product between $|U_b|$ and the similarity $sim(u, b)$. This way we weight the contribution of each bicluster with its size in addition to its similarity with the test user:

$$WF(i, b) = sim(u, b) * |U_b| \tag{4}$$

Finally, we apply the Most Frequent Item Recommendation (proposing those items that appear most frequently in the test user's formed neighborhood). Thus, we add the item weighted frequencies, we sort them, and propose the top-N items in the constructed list, which is customized to each test user preferences. The algorithm for the top-N generation list, is shown in Figure 6.

In our running example, assume that we keep all four biclusters (allow overlapping) and we are interested in 2 nearest biclusters ($k = 2$). As it is shown, U_9 has rated positively only two items (I_1,I_3). So, his similarity with each of the biclusters is (0.5, 0.5, 0, 0), respectively. Thus, test user's nearest neighbors come from the first two biclusters, and the recommended items for him will be items I_7 and I_5.

5 Experimental Configuration

In the sequel, we study the performance of the described nearest bicluster approach, against existing CF algorithms, by means of a thorough experimental evaluation. Henceforth, the proposed algorithm is denoted as Nearest Biclusters, the user-based algorithm as UB and the item-based algorithm as IB. Factors that are treated as parameters, are the following: the neighborhood size (k, default value 20), the size of the recommendation list (N, default value 20), and the size of training set and the test data set (default value 75% and 25%, respectively). The metrics we use are precision, recall, and F_1.

We performed experiments with several real data sets that have been used as benchmark in prior work. In particular, we examined two MovieLens data sets: (i) the first one with 100,000 ratings assigned by 943 users on 1,682 movies, denoted 100K data set (this is the default data set) and (ii) the second one with about 1 million ratings for 3,592 movies by 6,040 users, denoted 1M data set. The range of ratings is between 1(bad)-5(excellent) of the numerical scale, the P_τ threshold is set to 3 and the value of an unrated item is considered equal to zero. Moreover, we consider the division between not hidden and hidden data. For each transaction of a test user we keep the 75% as hidden data (the data we want to predict) and use the rest 25% as not hidden data (the data for modeling new users).

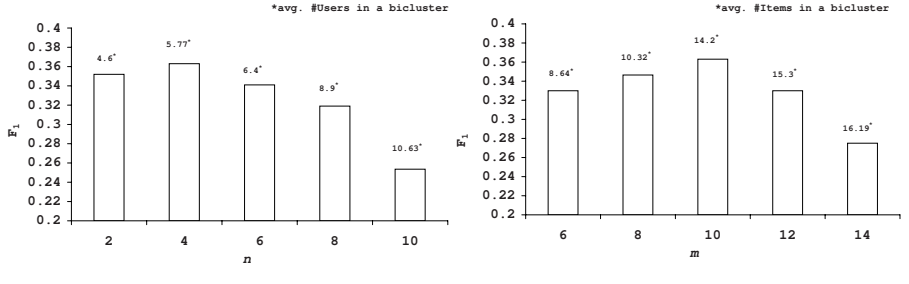
5.1 Results for Tuning Nearest Biclusters

As already discussed in Section 4.3, the only input of the Bimax algorithm is the minimum allowed number of users in a bicluster, n, and the minimum allowed number of items in a bicluster, m. In order to discover the best biclusters (in terms of effectiveness and efficiency), it is important to fine-tune these two input variables. So, we examine the performance of F_1 metric vs. different values for n and m.

Figure 7a illustrates F_1 for varying n (in this measurement we set $m = 10$). As n is the minimum allowed number of users in a bicluster, Figure 7a also depicts (through the numbers over the bars) the average numbers of users in a bicluster, which as expected increase with increasing n. As shown, the best performance is attained for $n = 4$. In the following, we keep this as the default value. Nevertheless, notice that performance is, in general, robust against varying n. In particular, for $n \leq 6$ the resulting F_1 is high. In contrast, for higher n, F_1 decreases. The reason is that with higher n we result with an inadequate number of biclusters to provide qualitative recommendations. The conclusion is that, small values for n are preferred, a fact that eases the tuning process.

Similarly, we examined F_1 for varying m. The results for F_1 are depicted in Figure 7b ($n = 4$). As previously, in the same figure we also illustrate the resulting average numbers of items in a bilcuster. The best performance is attained for $m = 10$ (henceforth kept as default value), whereas F_1 decreases for higher or lower m values. The reason is as follows: for very small values of m, there are not enough items in each bicluster to capture the similarity of users' preferences (i.e., matching is easily attained), thus the quality of recommendation decreases; on the other hand, for very large values of m, the number of discovered biclusters is not adequate to provide recommendations.

In Section 4.3, we mentioned that Bimax finds all biclusters that are not entirely contained in any other bicluster. It is obvious that this characteristic generates overlapping biclusters. The number of overlapping biclusters can be



(a) (b)

Fig. 7. F_1 vs. tuning number of (a) users, (b) items

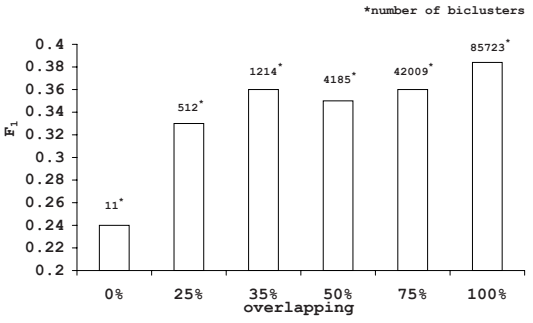


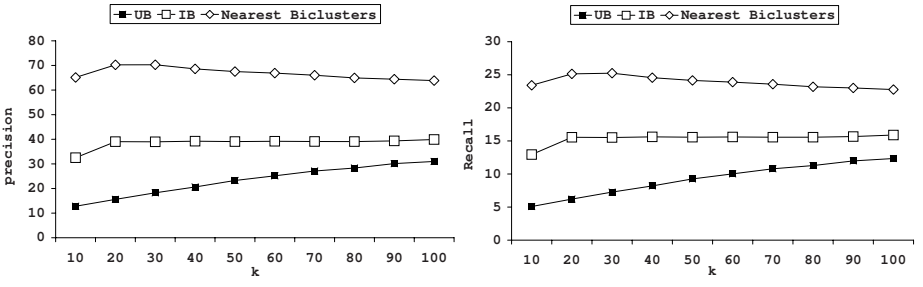
Fig. 8. F_1 vs. overlapping percentage between the biclusters

enormously large. To avoid this, we can perform a secondary filtering procedure to reduce the number of biclusters with respect to the desired overlapping degree. In Figure 8 we can see F_1 vs. varying overlapping degree (given as a percentage of common items/users between the biclusters). The figure also depicts (numbers over the bars) the resulting number of biclusters for each overlapping degree. With decreasing overlapping degree, F_1 decreases too. On the other hand, by keeping a high level of overlap between the biclusters, we harm efficiency –in terms of execution time– of the Nearest Biclusters algorithm (for its on-line part). As shown, by permitting 100% of overlapping, the number of generated biclusters is 85,723. It is obvious that this number impacts the efficiency of the recommendation process. The best combination of effectiveness and efficiency can be attained by having an overlapping equal to 35% (results to 1,214 biclusters), where the resulting F_1 is 0.36 (very close to the 100% overlapping result).

For the 1M data set, we follow the same tuning procedure and we resulted to the following values which have the best results in our experiments in terms of F1 measure : n=3, m=6, overlapping = 10%, which results to 2126 biclusters.

5.2 Comparative Results for Effectiveness

We now move on to the comparison of Nearest Bicluster algorithm with the UB and IB. The results for precision and recall vs. k are displayed in Figure 9a and b, respectively. As shown, the UB performs worst than IB for small values of k. The performance of the two algorithms converges to the same value as k increases. The reason is that with a high k, the resulting neighborhoods for both UB and IB are similar, since they include almost all items. Thus, the top-N recommendation lists are about the same, as they are formed just by the most frequent items. In particular, both UB and IB reach an optimum performance for a specific k. In the examined range of k values, the performance of UB and IB increases with increasing k and outside this range (not displayed), it stabilizes and never exceeds 40% precision and 15% recall.



(a) (b)

Fig. 9. Comparison between UB , IB and Bicluster CF in terms of (a) precision (b) recall

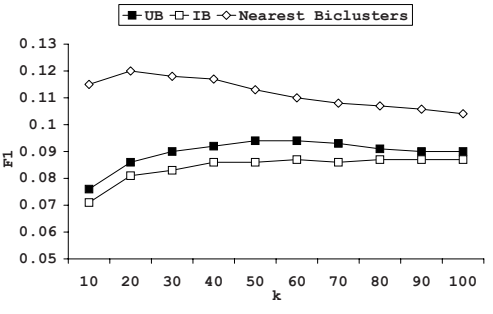


Fig. 10. Comparison between UB, IB, and Nearest Biclusters algorithm in terms of F1 measure for the 1M data set

Nearest Biclusters significantly outperforms UB and IB. The difference in precision is larger than 30%, whereas with respect to recall, it exceeds 10% (we refer to the optimum values resulting from the tuning of k). The reason is that Nearest Biclusters takes into account partial matching of preferences between users and the possibility of overlapping between their interests. In contrast, UB and IB are based on individual users and items, respectively, and do not consider the aforementioned characteristics.

For the 1M data set, the results for F1 measure vs. k are displayed in Figure 10. As shown, Nearest Biclusters outperforms UB and IB. The difference is analogous to the 100K data set. The reason again is that Nearest Biclusters takes into account partial matching of the users' preferences.

5.3 Comparative Results for Efficiency

Regarding efficiency, we measured the wall-clock time for the on-line parts of UB, IB and Nearest Biclusters algorithms. The on-line parts concern the time it takes to create a recommendation list, given what is known about a user. Notice

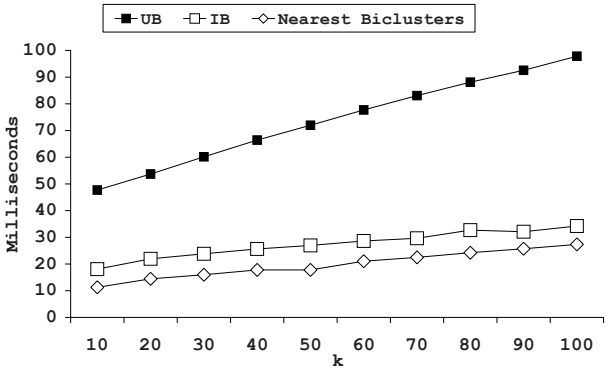


Fig. 11. Comparison between UB , IB and Nearest Biclusters in terms of execution time

that there is an off-line part for the IB and Nearest Biclusters, which demands additional computational time, needed to build the items' similarity matrix and find the biclusters, respectively. However, these computations are executed off-line. Thus, we do not count them in the recommendation time. Our experiments were performed on a 3 GHz Pentium IV with 1 GB of memory running the Windows XP operating system. The results vs. k are presented in Figure 11. In particular, we present the average time in milliseconds that takes to provide recommendations to a test user.

As shown, IB needs less time to provide recommendations than UB. Notice that the required time for IB to provide recommendation for a test user is almost stable, whereas the time for UB increases with increasing k. The reason is that UB finds, firstly, user neighbors in the neighborhood matrix and then counts presences of items in the user-item matrix. In contrast, with IB, the whole task is completed in the item-neighborhood matrix, which is generated off-line. Thus, in terms of execution time, IB is superior to UB.

In all cases, however, Nearest Biclusters needs even less time than IB to provide recommendations. This is due to the fact that the biclusters are also created off-line and, secondly, the number of biclusters in our experiment (1,214) is less than the number (1,682) of items' of the similarity matrix in the IB case. As it is already presented in Section 5.1, by decreasing the percentages of overlap between biclusters, accuracy decreases too. On the other hand, by keeping a high level of overlap between biclusters, we harm the efficiency. Thus, to combine effectiveness and efficiency, a prerequisite is a fine-tuning of the overlapping biclusters.

5.4 Examination of Additional Factors

In this section we examine the impact of additional factors. In our measurements we again consider UB, IB, and Nearest Biclusters algorithms.

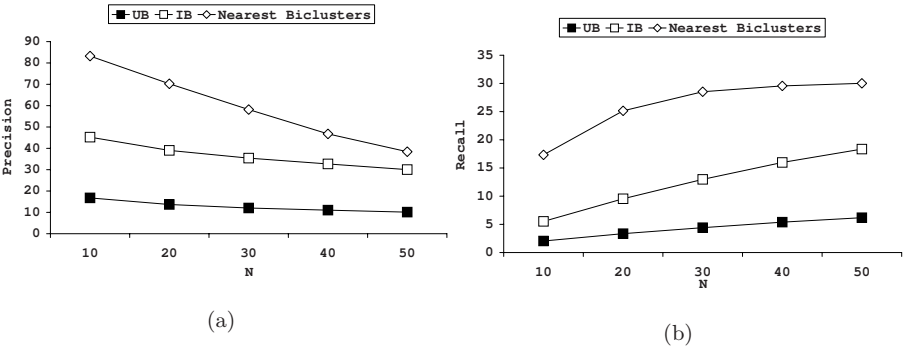


Fig. 12. Comparison vs. N: (a) precision, (b) recall

Recommendation list's size: We examine the impact of N. The results of our experiments are depicted in Figure 12. As expected, with increasing N, recall increases and precision decreases. Notice that the best performance of UB and IB corresponds to the worst performance of Nearest Biclusters. The relative differences between the algorithms are coherent with those in our previous measurements. We have to mentioned that in real applications, N should be kept low, because it is impractical for a user to see all recommendations when their number is large.

Training/Test data size: Now we test the impact of the size of the training set, which is expressed as percentage of the total data set size. The results for F_1 are given in Figure 13. As expected, when the training set is small, performance downgrades for all algorithms. Therefore, we should be careful enough when we evaluate CF algorithms so as to use an adequately large training sets. Similar to the previous measurements, in all cases Nearest Biclusters is better than IB and UB. The performance of both UB and IB reaches a peak around 75%,

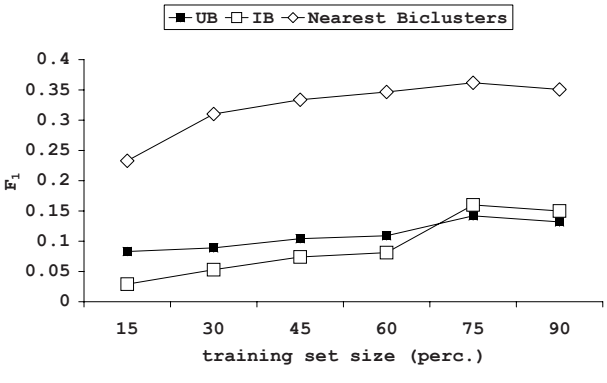


Fig. 13. Comparison vs. training set size

after which it reduces. It is outstanding that Nearest Biclusters trained with the 15% of the data set, attains much better F_1 than UB and IB when they are trained with 75%. Also, we see that after a threshold of the training set size, the increase in accuracy for algorithms is less steep. However, the effect of overfitting is less significant compared to general classification problems. In contrast, low training set sizes negatively impact accuracy. Therefore, the fair evaluation of CF algorithms should be based on adequately large training sets.

6 Conclusions

We proposed the application of an exact biclustering algorithm in the CF area, to disclose the duality between users and items and to capture the range of the user's preferences. In addition, we propose a novel nearest-biclusters algorithm, which uses a new similarity measure that achieves partial matching of users' preferences and allows overlapping interests between users.

We performed experimental comparison of the nearest biclusters algorithm against well known CF algorithms, like user-based or item-based methods. Our extensive experimental results illustrate the effectiveness and efficiency of the proposed algorithm over the existing approaches.

We highlight the following conclusions from our examination:

- Our approach shows significant improvements over existing CF algorithms, in terms of effectiveness, because it exploits the duality of users and items through biclustering and partial matching of users' preferences. In particular, we attain more than 30% improvement and recall more than 10% in terms of precision and recall, respectively.
- Our approach shows improvements over existing CF algorithms, in terms of efficiency. The Nearest Biclusters algorithm needs even less time than item based approach to provide recommendations.
- In our experiments we have seen that only a 15% of the training set is adequate to provide accurate results.
- We introduced a similarity measure for the biclusters' neighborhood formation and proposed the Weighted Frequency for the generation of the top-N recommendation list of items.

Summarizing the aforementioned conclusions, we see that, the proposed Nearest Biclusters algorithm through a simple, yet effective, biclustering algorithms (Bimax) and the partial matching of users' preferences, achieves better results in terms of effectiveness and efficiency than traditional CF algorithms. For this reason, in our future work we will compare biclusters with constant values with other categories of biclusters (i.e biclusters with coherent values). The first category looks for subsets of rows and subsets of columns with constant values, while the second is interested in biclusters with coherent values. Moreover, we will also

examine different similarity measures between a user and a bicluster based on the items that are included in the bicluster or based on features characteristics of those items.

References

1. Ben-Dor, A., Chor, B., Karp, R., Yakhini, Z.: Discovering local structure in gene expression data: The order-preserving submatrix problem. Journal of Computational Biology 10(3/4), 373–384 (2003)
2. Cheng, Y., Church, G.: Biclustering of expression data. In: Proceedings of the ISMB Conference, pp. 93–103 (2000)
3. Deshpande, M., Karypis, G.: Item-based top-n recommendation algorithms. ACM Transactions on Information Systems 22(1), 143–177 (2004)
4. Dhillon, I.S.: Co-clustering documents and words using bipartite spectral graph partitioning. In: Proceedings of the ACM SIGKDD Conference (2001)
5. Dhillon, I.S., Mallela, D.S., Modha, S.: Information theoretic co-clustering. In: Proceedings of the ACM SIGKDD Conference (2003)
6. Goldberg, D., Nichols, D., Brian, M., Terry, D.: Using collaborative filtering to weave an information tapestry. ACM Communications 35(12), 61–70 (1992)
7. Hartigan, J.A.: Direct clustering of a data matrix. Journal of the American Statistical Association 67(337), 123–129 (1972)
8. Herlocker, J., Konstan, J., Borchers, A., Riedl, J.: An algorithmic framework for performing collaborative filtering. In: Proceedings of the ACM SIGIR Conference, pp. 230–237 (1999)
9. Herlocker, J., Konstan, J., Riedl, J.: An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. Information Retrieval 5(4), 287–310 (2002)
10. Herlocker, J., Konstan, J., Terveen, L., Riedl, J.: Evaluating collaborative filtering recommender systems. ACM Transactions on Information Systems 22(1), 5–53 (2004)
11. Hofmann, T., Puzicha, J.: Latent class models for collaborative filtering. In: Proceedings of the IJCAI Conference (1999)
12. Ihmels, J., Bergmann, S., Barkai, N.: Defining transcription modules using large-scale gene expression data. Bioinformatics 20(13), 1993–2003 (2004)
13. Karypis, G.: Evaluation of item-based top-n recommendation algorithms. In: Proceedings of the ACM CIKM Conference, pp. 247–254 (2001)
14. Long, B., Zhang(Mark), Z., Yu, P.S.: Co-clustering by block value decomposition. In: KDD 2005. Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining, pp. 635–640. ACM Press, New York (2005)
15. Madeira, S., Oliveira, A.: Biclustering algorithms for biological data analysis: a survey. ACM Transactions on Computational Biology and Bioinformatics 1, 24–45 (2004)
16. McLauglin, R., Herlocher, J.: A collaborative filtering algorithm and evaluation metric that accurately model the user experience. In: Proceedings of the ACM SIGIR Conference, pp. 329–336 (2004)
17. Mirkin, B.: Mathematical classification and clustering. Kluwer Academic Publishers, Dordrecht (1996)

18. Mobasher, B., Dai, H., Luo, T., Nakagawa, M.: Improving the effectiveness of collaborative filtering on anonymous web usage data. In: Proceedings of the Workshop Intelligent Techniques for Web Personalization, pp. 53–60 (2001)
19. Murali, T., Kasif, S.: Extracting conserved gene expression motifs from gene expression data. In: Proceedings of the Pacific Symposim on Biocompomputing Conference, vol. 8, pp. 77–88 (2003)
20. Prelic, A., et al.: A systematic comparison and evaluation of biclustering methods for gene expression data. Technical Report (2005)
21. Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., Riedl, J.: Grouplens: An open architecture for collaborative filtering on netnews. In: Proceedings of the Computer Supported Collaborative Work Conference, pp. 175–186 (1994)
22. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Analysis of recommendation algorithms for e-commerce. In: Proceedings of the ACM Electronic Commerce Conference, pp. 158–167 (2000)
23. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Application of dimensionality reduction in recommender system-a case study. In: ACM WebKDD Workshop (2000)
24. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item-based collaborative filtering recommendation algorithms. In: Proceedings of the WWW Conference, pp. 285–295 (2001)
25. Symeonidis, P., Nanopoulos, A., Papadopoulos, A., Manolopoulos, Y.: Collaborative filtering process in a whole new light. In: Proc. IDEAS conf., pp. 29–36 (2006)
26. Tanay, A., Sharan, R., Kupiec, M., Shamir, R.: Revealing modularity and organization in the yeast molecular network by integrated analysis of highly heterogeneous genomewide data. In: Proceedings of the National Academy of Science conference, pp. 2981–2986 (2004)
27. Tanay, A., Sharan, R., Shamir, R.: Discovering statistically signifnicant biclusters in gene expression data. In: Proceedings of the ISMB conference (2002)
28. Ungar, L., Foster, D.: A formal statistical approach to collaborative filtering. In: Proceedings of the CONALD Conference (1998)
29. Xue, G., Lin, C., Yang, Q., et al.: Scalable collaborative filtering using cluster-based smoothing. In: Proceedings of the ACM SIGIR Conference, pp. 114–121 (2005)