Incremental Collaborative Filtering via Evolutionary Co-clustering

Mohammad Khoshneshin
Management Sciences Department
University of Iowa
Iowa City, IA 52242 USA
mohammad-khoshneshin@uiowa.edu

W. Nick Street
Management Sciences Department
University of Iowa
Iowa City, IA 52242 USA
nick-street@uiowa.edu

ABSTRACT

Collaborative filtering is a popular approach for building recommender systems. Current collaborative filtering algorithms are accurate but also computationally expensive, and so are best in static off-line settings. It is desirable to include the new data in a collaborative filtering model in an online manner, requiring a model that can be incrementally updated efficiently. Incremental collaborative filtering via co-clustering has been shown to be a very scalable approach for this purpose. However, locally optimized co-clustering solutions via current fast iterative algorithms give poor accuracy. We propose an evolutionary co-clustering method that improves predictive performance while maintaining the scalability of co-clustering in the online phase.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data Mining

General Terms

algorithms, theory.

Keywords

Incremental collaborative filtering, co-clustering, evolutionary algorithm, ensembles.

1. INTRODUCTION

Recommender systems suggest items of interest to users. *Collaborative filtering* (CF) uses purchase or rating information to recommend items based on similarity [3]. If two users have liked (or disliked) similar items up to now, it is likely that they will have the same behavior in the future. Therefore, collaborative filtering systems recommend based on the history of ratings. One of the drawbacks in current CF approaches is that they are more appropriate for static settings; incorporating new data to the model may be a non-trivial task. In real world problems, there are always new users and items that should be incorporated into the model recommendations in an online manner. Incremental collaborative algorithms are intended to handle this need.

A few published approaches have addressed the incremental CF problem. Sarwar et al. [7] and Brand [2] proposed using singular value decomposition as an online CF strategy. Das et al. [4] proposed a scalable online CF approach using MinHash clustering, Probabilistic Latent Semantic Indexing (PLSI), and co-visitation counts. However in this work, only binary ratings (such as implicit user feedback) were considered and therefore the proposed approaches are not applicable to prediction problems. In K-nearest neighbors collaborative filtering approaches, similarity parameters such as correlation can be updated incrementally during the online phase [6].

George and Merugu [5] used co-clustering as a scalable incremental CF approach for dynamic settings. They showed the performance of incremental co-clustering is comparable to incremental SVD but much more scalable. For implementing co-clustering they used the approach of Bregman co-clustering [1], a very fast iterative local search which can result in very poor local optima.

In this paper we propose an incremental CF method that is both scalable and accurate. We use an evolutionary co-clustering algorithm that finds better solutions than Bregman co-clustering at the cost of offline training time, which is relatively unimportant. Also, we revise the incremental algorithm suggested in [5] and introduce an ensemble strategy to give better predictions.

2. INCREMENTAL CF

In a collaborative filtering problem, there are U users and V items. Users have provided a number of explicit ratings for items; r_{ui} is the rating of user u for item i. There are two phases in a CF algorithm: an offline phase in which training based on known ratings is performed, and an online phase in which unknown ratings are estimated using the output of the offline phase. Most CF approaches use only the data available offline to predict ratings. In incremental CF, the data available during online phase is incorporated into future predictions, potentially improving the predictive accuracy.

2.1 Baseline algorithm

The simplest way to predict a rating is the global average of all ratings. However, some users tend to rate higher and some items are more popular. Including user bias and item bias in rating, we can predict user ratings by

$$\hat{r}_{ui} = (1 - S_{n_u,\omega} - S_{n_i,\omega})\bar{r} + S_{n_u,\omega}\bar{r}_u + S_{n_i,\omega}\bar{r}_i \qquad (1)$$

where $\bar{r}$ is the global average, $\bar{r}_u$ is the average of ratings by user u, $\bar{r}_i$ is the average of ratings for item i, n_u is the number of ratings by user u, and n_i is the number of ratings for item i. $S_{n_u,\omega}$ and $S_{n_i,\omega}$ are the support function for user u and item i which we define as

$$S_{\upsilon,\omega} = \begin{cases} \frac{\upsilon}{\omega} & if \quad \upsilon < \omega, \\ 1 & otherwise. \end{cases} \tag{2}$$

If fewer ratings are available for a user or item, the support will be smaller. The parameter ω determines the necessary support. Empirically we found three to be a robust choice for ω. When the support of a user and an item is zero, then $\hat{r}_{ui} = \bar{r}$; when the support is one, $\hat{r}_{ui} = \bar{r}_u + \bar{r}_i - \bar{r}$; and when the support of a user u is zero and an item i is one, then $\hat{r}_{ui} = \bar{r}_i$, and vice versa for the reverse.

2.2 Incremental CF via co-clustering

Clustering refers to partitioning similar objects into groups [1]. Co-clustering partitions two different kinds of objects simultaneously. If one views the clustering problem as grouping rows of a matrix together, then co-clustering is the simultaneous grouping of rows and columns.

In collaborative filtering via co-clustering as suggested in [5], each user u is assigned to a user cluster (represented by $\rho(u)$) and each item i is assigned to an item cluster (represented by $\gamma(i)$) and the prediction is as follows:

$$\hat{r}_{ui} = \bar{r}_{kl} + (\bar{r}_u - \bar{r}_k) + (\bar{r}_i - \bar{r}_l), \tag{3}$$

where $k = \rho(u)$ is the user cluster assigned to user u, $l = \gamma(i)$ is the item cluster assigned to item i, $\bar{r}_{kl}$ is the average of ratings belonging to users in user cluster k and items in item cluster l, $\bar{r}_k$ is the average of ratings belonging to users in user cluster k, and $\bar{r}_l$ is the average of ratings belonging to items in item cluster l. The term $(\bar{r}_u - \bar{r}_k)$ tries to remove the bias of user u (some users tend to rate higher) and $(\bar{r}_i - \bar{r}_l)$ is the same for item i (some items are more likable).

George and Merugu [5] used a fast iterative heuristic proposed by Banerjee et al. [1]. This algorithm has two phases: updating user clusters and updating item clusters. When updating user clusters, we assume all co-cluster means are constant and all items are assigned to item clusters, then we assign each user so that the sum of squared errors is minimized. Item cluster updating follows the same approach. Details can be found in [1, 5].

In the online phase, the prediction is as follows:

$$\hat{r}_{ui} = \begin{cases} \bar{r}_{kl} + (\bar{r}_u - \bar{r}_k) + (\bar{r}_i - \bar{r}_l) & if\ (oldUser\text{-}oldItem) \\ \bar{r}_i & if\ (newUser\text{-}oldItem) \\ \bar{r}_u & if\ (\ oldUser\text{-}newItem) \\ \bar{r} & if\ (newUser\text{-}newItem) \end{cases} \tag{4}$$

In [5], incremental training is achieved by using new ratings to update the average parameters ($\bar{r}_{kl}$, $\bar{r}_u$, $\bar{r}_k$, $\bar{r}_i$, $\bar{r}_l$) in equation (3). However, new users or items are not assigned to clusters during the online phase.

3. INCREMENTAL EVOLUTIONARY CO-CLUSTERING

As our experimental results will show, incremental collaborative filtering via co-clustering [5] discussed in Section 2.2 may be even worse than the baseline. Therefore, we propose a number of revisions to improve this method.

First, if the support (number of available ratings) for a user or item is low, the co-clustering approach will not provide good predictions for them, and the training phase will be affected by these noisy inputs. As a strategy, users and items with low support are eliminated from the training phase so that training is both more effective and efficient.

The prediction equation (3) is not appropriate for the incremental scenario, because it incorporates three average parameters ($\bar{r}_{kl}$, $\bar{r}_k$, $\bar{r}_l$) from a co-clustering solution that is not necessarily reliable. This is due to the fact that $\bar{r}_k$ and $\bar{r}_l$ tend to be very close to the global average empirically. Therefore, this model can be equivalent to $\hat{r}_{ui} = \bar{r}_{kl} + \bar{r}_u + \bar{r}_i - 2\bar{r}$ during the online phase. On the other hand, using only the block average $\bar{r}_{kl}$ for prediction ignores user and item bias which results in poor accuracy as well. To overcome this problem, we begin by co-clustering residuals, rather than ratings. We model a rating prediction as

$$\hat{r}_{ui} = \bar{r}_u + \bar{r}_i - \bar{r} + \varepsilon_{ui}. \tag{5}$$

Note that $\bar{r}_u + \bar{r}_i - \bar{r}$ is the same as the prediction equation (1) when the support function is 1. As a result, ε_{ui} is the correction parameter for (1). For a known rating, (5) can be rewritten as

$$\varepsilon_{ui} = r_{ui} - (\bar{r}_u + \bar{r}_i - \bar{r}) \tag{6}$$

where ε_{ui} can be interpreted as the residual of the prediction via (1). For implementing co-clustering, it is enough to work with the following objective function:

$$\min_{\rho,\gamma} \sum_u \sum_i w_{ui} \left(\varepsilon_{ui} - \bar{\varepsilon}_{\rho(u)\gamma(i)}\right)^2 \tag{7}$$

where w_{ui} is one if rating r_{ui} exists in training dataset and otherwise is zero. $\bar{\varepsilon}_{\rho(u)\gamma(i)}$ is the block average of residuals for user cluster $\rho(u)$ and item cluster $\gamma(i)$.

Now we can define the prediction strategy based on (1) and (5). For old user - old item,

$$\hat{r}_{ui} = \bar{r}_u + \bar{r}_i - \bar{r} + \bar{\varepsilon}_{\rho(u)\gamma(i)} \tag{8}$$

and otherwise

$$\hat{r}_{ui} = (1 - S_{n_u,\omega} - S_{n_i,\omega})\bar{r} + S_{n_u,\omega}\bar{r}_u + S_{n_i,\omega}\bar{r}_i. \tag{9}$$

As mentioned, one advantage of co-clustering is scalability. Therefore creating an ensemble of different co-clusterings is desirable. Ensembles are used to improve the accuracy of a method using a group of predictors, while increasing the running time linearly with the number of ensemble elements. Let p denote a co-clustering solution and P be the number of co-clusterings we use in the model. We can predict with

$$\hat{r}_{ui} = \bar{r}_u + \bar{r}_i - \bar{r} + \frac{\sum_p \exp(-z_{ulp} - z_{ikp})\bar{\varepsilon}_{klp}}{\sum_p \exp(-z_{ulp} - z_{ikp})} \tag{10}$$

where z_{ulp} is the average error of prediction for user u and item cluster l in co-clustering solution p and similarly z_{ikp} is the average error of prediction for item i and user cluster k in co-clustering solution p. Intuitively if previous predictions of a co-clustering solution are better, its weight is higher.

In the algorithm proposed in [5], new users and items are not included in the co-clustering during the online phase. As a result, the model is unable to provide legitimate predictions for them. However, in the revised version, it is trivial to find an appropriate cluster for a user or item. Let u be a

```
Input:  new rating, co-clustering ($\rho$,$\gamma$)
Update user average $\bar{r}_u$
Update item average $\bar{r}_i$
IF {old user - old item}
   Update co-cluster mean $\bar{\varepsilon}$ and weights $z$
ELSEIF {old user - new item} AND {numberIn(item)>$\tau$}
   Assign the item to an item cluster
   Update co-cluster mean $\bar{\varepsilon}$ and weights $z$
ELSEIF {new user - old item} AND {numberIn(user)>$\tau$}
   Assign the user to a user cluster
   Update co-cluster mean $\bar{\varepsilon}$ and weights $z$
ENDIF
```

Figure 1: Incremental training in evolutionary co-clustering

new user who has provided some ratings. If a sufficient number of rated items exist in the current co-clustering solution, then the new user's cluster can be found using [1]:

$$\rho(u) = \arg\min_g \sum_h n_{uh}(\bar{\varepsilon}_{uh} - \bar{\varepsilon}_{gh})^2 \tag{11}$$

where n_{uh} is the number of times user u has rated the items belonging to item cluster h during the online phase, and $\bar{\varepsilon}_{uh}$ is the average of residuals for those ratings. A similar procedure finds the cluster of a new item. Figure 1 shows the incremental training algorithm. The function *numberIn*() is the number of ratings a user (item) has in the co-clustering solution which is defined by $\sum_h n_{uh}$ ($\sum_g n_{ig}$) for user u (item i). If this value is bigger than a threshold τ, then we can trust the information to incorporate the new user or item. Otherwise, the risk of misclustering will be high. Subsequently, new users and items will not receive prediction from co-clustering at the very beginning. These ratings will be estimated using (1), which is more accurate experimentally. As a rule of thumb, τ can be set to 3 since we wish to incorporate new users or items as soon as possible but not at the cost of bad predictions.

We now turn to the construction of co-clusterings via evolutionary algorithms, a population-based search approach that explores a solution space by evolving a group of individuals to find good solutions. In our context, let P co-clustering solutions exist. The goal is to find better solutions by combining the current solutions. Every evolutionary algorithm has three main steps: selection, in which two or more individuals are chosen to create offspring; crossover, in which the selected items are combined to create new solutions; and replacement, in which the new solutions replace existing solutions if they satisfy some criteria.

Our evolutionary co-clustering algorithm is shown in Figure 2. A group of co-clustering solutions is randomly generated and locally optimized via iterative Bregman co-clustering. The numbers of user and item clusters are randomly selected from a specific range. In the evolutionary iteration phase, two co-clustering solutions are randomly selected for crossover. In the context of optimization, better individuals are chosen with higher probability. However, in machine learning, generalization is more important than optimization and biased selection may result in premature convergence. Our initial studies indicated that randomly selecting co-clusterings improved generalization. After selecting two co-clusterings, a new solution is generated via the crossover function presented in Figure 3.

The crossover operation is between two clusters ϕ_1 and ϕ_2. If the required number of clusters is K, the $K-1$ largest

```
Input:  Population size $P$, Rating matrix

Initialization:
  FOR $p = 1$ to $P$
        $K_p \leftarrow RandomInteger(1, \beta_1)$
        $L_p \leftarrow RandomInteger(1, \beta_2)$
        $\forall u:\ \rho_0(u) \leftarrow RandomInteger(1, K_p)$
        $\forall i:\ \gamma_0(i) \leftarrow RandomInteger(1, L_p)$
        $(\rho_p, \gamma_p) \leftarrow LocallyOptimizationCooclustering(\rho_0, \gamma_0)$
   ENDFOR
Evolutionary iteration:
Repeat
   Select 2 co-clusterings $(\rho_q, \gamma_q)$ and $(\rho_r, \gamma_r)$ randomly
   $K_o \leftarrow \frac{K_q+K_r}{2} + RandomInteger(-\beta_3, \beta_3)$
   $L_o \leftarrow \frac{L_q+L_r}{2} + RandomInteger(-\beta_4, \beta_4)$
   $\rho_o \leftarrow crossover(\rho_q, \rho_r)$
   $\gamma_o \leftarrow crossover(\gamma_q, \gamma_r)$
   $(\rho_o, \gamma_o) \leftarrow LocallyOptimizationCooclustering(\rho_o, \gamma_o)$
   DiscardWorst $\{(\rho_p, \gamma_p)_{p=1}^{P}, (\rho_o, \gamma_o)\}$
Until{convergence OR iter>maxIter}
```

Figure 2: Evolutionary algorithm

```
Input:  clusters $\phi_1$ and $\phi_2$,required cluster size $K$
Output:  offspring cluster $\phi_o$

$\forall x, q, r : \zeta_{qr} \leftarrow x|\phi_1(x) = q\ \&\ \phi_2(x) = r$
FOR $k = 1$ to $K - 1$
    $\forall x \in \zeta^{(k)}:\ \phi_o(x) \leftarrow k$
ENDFOR
$\forall x \notin \cup_{k=1}^{K-1}\zeta^{(k)}:\ \phi_o(x) \leftarrow K$
```

Figure 3: Crossover algorithm. ζ_{qr} is the intersection between cluster q in clustering ϕ_1 and cluster r in clustering ϕ_2. $\zeta^{(k)}$ is the k^{th} largest intersection.

intersections between ϕ_1 and ϕ_2 are assigned to the first $K-1$ clusters and the remainder will be assigned to the last cluster. Formally, let X be a $N \times K$ assignment matrix in which an element (u, k) is one if object u is assigned to cluster k and zero otherwise. Then the intersection matrix can be defined as $X'X$.

In the next step, the offspring from crossover will be locally optimized via the fast iterative Bregman co-clustering algorithm from [1]. This is an important step due to the fact that most objects will be assigned to the last cluster. However, since the iterative co-clustering first estimates averages and then assigns users and items, we hope that most of the blocks will have good quality averages via fewer but selected users or items.

Finally, we should either discard a current solution or the offspring to preserve the total number of solutions. The following function can be used to discard the worst solution:

$$\widetilde{p} = arg\max_p \sum_u \sum_i w_{ui}\left(\varepsilon_{ui} - \frac{\sum_{p' \neq p} \bar{\varepsilon}_{\rho_{p'}(u)\gamma_{p'}(i)}}{P}\right).$$

where $\widetilde{p}$ represents the worst solution in the population. If the worst solution is the offspring we just discard it, otherwise the worst solution will be replaced with the offspring.

4. EXPERIMENTAL RESULTS

In this section, we present the results of experiments performed to evaluate the effectiveness of our method. We used the Movielens dataset[1] consisting of 100,000 ratings (1-5) by 943 users on 1682 movies. We used mean absolute error

[1]http://www.grouplens.org/data/

	20%-80%	50%- 50%	80%-20%
ECOCLE	.7563	.7284	.7155
ECOCL	.7626	.7433	.7336
Baseline	.7645	.7586	.7555
COCL	.7781	.7560	.7513
IKNN	.7646	.7489	.7436
SVD	$> .79$	$> .75$	$> .73$

Table 1: Average MAE of different methods

(MAE) to evaluate and compare different methods. Four methods were used for comparison:

1. Baseline: based on the model proposed in Section 3.1.
2. COCL: The method presented in [5].
3. ECOCL: Evolutionary co-clustering without ensembles.
4. ECOCLE: Evolutionary co-clustering with ensembles.
5. IKNN: Incremental KNN method [6].
6. SVD: We compared our results with those from [7] for SVD on the same dataset and similar settings.

In our experiments, we used 5-fold cross-validation. First a part of data was held for offline training. Then the rest of data was included in an online phase which is a combination of incremental training and prediction. In the online phase, first a predicted rating was used for computing prediction error, and then the new case was incorporated into the model. The sequence of data was randomized. We performed incremental training based on three different strategies: "20%-80%", in which 20% of data was used for training and 80% for incremental training; "50%-50%," and "80%-20%."

All of the support parameters such as ω in (1) were set to 3. For the COCL method, we implemented the algorithm for different user and item cluster numbers and the best result (10 user clusters and 2 item clusters) is reported. The number of ensembles for ECOCLE was set to 25 and the iteration limit was 250.

The results for all methods are summarized in Table 1. First, the baseline is reasonably good compared to other methods when less data is available during the training phase. As more data is provided for the offline phase, other methods are more accurate than the baseline. Also, evolutionary co-clustering algorithm (ECOCLE) is more successful when more data is available. Using ECOCLE in all phases gives the best results. Evolutionary co-clustering without ensembles (ECOCL) still outperforms other methods while its performance is slightly better than baseline for the 20%-80% case. We did not perform SVD and only report the result of Sarwar et al. [7]. For 20%-80%, SVD has the poorest performance. However, as more data is available for training, it gets more competent. Note that the experimental protocol of [7] was different than ours in that new users and items were incrementally added to the model in one step, based on a training/test split. However, it is non-trivial to update an SVD model based on new data. Therefore, the performance of incremental SVD in our protocol might be similar.

The time of both offline and online training is reported in the Table 2. The offline phase of ECOCLE needs more time due to the evolutionary algorithm. However, since this phase only needs to be done once, greater offline training time can be ignored. Online time is the sum of both incremental online training and online prediction. ECOCLE and IKNN have similar online speeds, while the accuracy for ECOCLE is much higher. The time problem could be mitigated by parallelizing the co-clustering operations, since updating is independent for the different solutions in the ensemble.

	offline	online
ECOCLE	3.350	1.872
ECOCL	1.783	.008
Baseline	.000	.006
COCL	.008	.009
IKNN	.053	1.532

Table 2: Average time (milliseconds) of different methods per rating. The online time is the sum of online prediction and incremental training

5. CONCLUSION AND DISCUSSION

Online collaborative filtering methods that can incorporate new data in real time are advantageous in many practical situations. However, this problem has not been adequately addressed. In this paper we extended the idea of CF via co-clustering to fulfill this need. As our empirical results showed, our method achieved very good accuracy compared to other incremental methods. Training was comparatively slow, but still manageable, and could be improved by a straightforward parallelization. Further, most of the algorithmic parameters of our method were chosen randomly from a wide range of values and no fine-tuning was done to find a better range. Our method provides a recommender system that gives accurate results and updates incrementally without spending time on tuning.

6. REFERENCES

[1] A. Banerjee, I. Dhillon, J. Ghosh, S. Merugu, and D.S. Modha. A generalized maximum entropy approach to Bregman co-clustering and matrix approximation. *Journal of Machine Learning Research*, 8:1919–1986, 2007.

[2] M. Brand. Fast online SVD revisions for lightweight recommender systems. In *SIAM Intl. Conf. on Data Mining*, pages 37–46, 2003.

[3] L. Candillier, F. Meyer, and M. Boulle. Comparing state-of-the-art collaborative filtering systems. *Lectures Notes in Computer Science*, 4571:548, 2007.

[4] A.S. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: Scalable online collaborative filtering. In *Intl. Conf. on the World Wide Web*, pages 271–280. ACM New York, NY, USA, 2007.

[5] T. George and S. Merugu. A scalable collaborative filtering framework based on co-clustering. In *IEEE Intl. Conf. on Data Mining*, pages 625–628, 2005.

[6] M. Papagelis, I. Rousidis, D. Plexousakis, and E. Theoharopoulos. Incremental collaborative filtering for highly-scalable recommendation algorithms. In *Intl. Symp. on Methodologies of Intelligent Systems*, pages 553–561, 2005.

[7] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Incremental singular value decomposition algorithms for highly scalable recommender systems. In *Intl. Conf. on Computer and Information Science*, pages 27–28, 2002.