A cluster-based matrix-factorization for online integration of new ratings

Modou Gueye
Université Cheikh Anta DIOP
BP. 16432 Dakar Fann, Sénégal

Talel Abdessalem
Institut Telecom - Telecom ParisTech
46, rue Barrault 75013 Paris, France

Hubert Naacke
LIP6 - UPMC Sorbonne Universités
4 place Jussieu 75005 Paris, France

May 23, 2011

Abstract

Les techniques de factorisation de matrice offrent une bonne qualité de prédiction pour les systèmes de recommandation, ce qui explique l'engouement qu'elles succitent depuis quelques années. Cependant, leur coût de calcul élevé constitue un obstacle à leur utilisation dans des contextes dynamiques, où les avis des utilisateurs sur des produits sont générés de façon continue. L'integration de nouvelles évaluations de produits est nécéssaire pour maintenir une bonne qualité de prédiction. Mais, ceci nécessite une refactorisation de la matrice à chaque fois que de nouvelles évaluations sont disponibles, ce qui est impossible de faire. Ce papier propose une solution qui permet de ralentir la dégradation des prédictions entre deux factorisations, en gérant des biais par utilisateur. La prise en compte de nouvelles évaluations se fait au niveau des biais avec un coût assez faible. Les expériences menées sur des jeux de données réels confirment notre analyse et montrent l'efficacité de notre solution.

It is today accepted that matrix factorization models allow a high quality of rating prediction in recommender systems. However, a major drawback of matrix factorization is its static nature that results in a progressive declining of the accuracy of the predictions after each factorization. This is due to the fact that the new obtained ratings are not taken into account until a new factorization is computed, which can not be done very often because of the high cost of matrix factorization. In this paper, aiming at improving the accuracy of recommender systems, we propose a cluster-based matrix factorization technique that enables online integration of new ratings. Thus, we significantly enhance the obtained predictions between two matrix factorizations. The experiments we did on a large dataset demonstrated the efficiency of our approach. Furthermore, we devise an experimental protocol that allows for identifying the optimal number of ratings to take into account for factorization. This contributes as a practical solution addressing the tradeoff between taking into account more ratings into factorization for better quality, versus missing fewer ratings during factorization.

1 Introduction

Due to their great commercial value, recommender systems attract a lot of attention [11, 15, 18, 12]. Their purpose is to predict user preferences on a huge selection of items, i.e. find items that are likely to be of interest for the user. Because the user is often overwhelmed for facing the considerable amount of items provided by electronic retailers, the predictions are a salient function of all types of e-commerce [26, 5]. Collaborative filtering is a widely used category of recommender systems. It consists in analyzing relationships between users and interdependencies among items to identify new user-item associations [27, 17, 23]. Based on these associations, recommendations are inferred.

One of the most successful collaborative filtering algorithms is matrix factorization (MF), which has proved its good scalability and predictive accuracy [29, 16]. In its basic form, matrix factorization profiles both items and users by vectors of factors inferred from rating patterns. High correspondence between item and user factors leads to a recommendation. Although matrix factorization is very popular because of its proven qualities, some shortcomings remain. One of these is the fact that the model generated by MF is static. Once it has been generated the model delivers recommendations based on a snapshot of the incoming ratings frozen at the beginning of the generation. To take into account the missing ratings (those arrived after the last model generation), the model have to be computed periodically. However, it is not realistic to carry it out frequently, because of the high cost of model recomputation. Therefore, the quality of the recommendations will decrease gradually until a new model is computed.

In real-world context where new ratings happen continuously, users profile evolve dynamically. Consider, for instance, a costumer of an online music-store looking for good pop songs. He asks the application for some recommendations and the system proposes to him a short list of songs with high probability of interest (based on the latest available model). The costumer selects and rates the songs he already knows or he just listened to, and asks for new recommendations. Since the preferences of the customers evolve accordingly to the songs they have listened to, it is important to be able to integrate the new ratings for the subsequent recommendations (including those provided for the customers having a similar profile). Otherwise, the following recommendations will be unnecessary.

In this paper, we propose a solution that reduces the mitigation of the quality of the recommendations over time. It combines clustering, matrix factorization and bias adjustment [23, 28], in order to startup with a high quality model. The biases are continuously updated with the new ratings, to maintain a satisfactory quality of recommendations for a longer time. Our solution is based on the observation that the rating tendency of a user is not uniform, and can change from one set of items to another. A list of biases is then associated to each user, one bias for each set of items holding some similarities in common. Thus, the integration of a new rating is provided by recomputing a local user bias (a bias of a user for a specific cluster of items), which may be done very quickly. Our approach improves the scalability of recommender systems by reducing the frequency of model recomputations. The experiments we conducted on the MovieLens dataset [3] confirmed that our technique is well adapted for dynamic environments where ratings happen continuously. The cost of the integration of new ratings is very low, and the quality of our recommendations doesn't decrease very fast between two successive matrix factorizations.

The remainder of this paper is organized as follows. In Section 2 we present some preliminary notions and requirements. Section 3 details our cluster-based matrix fac-

torization solution. In Section 4, we present an experimental analysis of our proposal. Section 5 summarizes the related work, and Section 6 concludes the paper.

2 Preliminaries

The principal purpose of recommender systems is to predict the interest of a user for a given item, e.g. to determine how much the user would like the item. Most of the time, this interest is represented by numerical values from a fixed range (one to five stars, for example).

2.1 Prediction issue

Consider a set U of users and a set I of items. User ratings can be seen as tuples (u, i, r_{ui}, t_{ui}), where u denotes a user, i denotes an item, r_{ui} the rating of user u for the item i, and t_{ui} is a timestamp. We assume that a user rates an item at most once.

The problem is to predict the future ratings such that the difference between an estimated rating $\hat{r}_{ui}$ and its true value r_{ui} is the lowest possible. In order to build the estimator, the set of existing ratings is split in two parts: the first part is used for the training step and the second part for the evaluation of the accuracy of the estimator.

The quality of a recommender system can be decided on the accuracy of its predictions. The Mean Absolute Error (MAE), which computes the average of the absolute difference between the predictions and true ratings, is the most widely used metric for the evaluation of recommender systems [14, 27]. In this paper we use the MAE metric to compare our proposition to traditional systems.

$$MAE = \frac{1}{n} \sum_{u,i} |r_{ui} - \hat{r}_{ui}| \tag{1}$$

where n is overall number of ratings. The lower the MAE, the better is the prediction.

2.2 Matrix factorization

In the recommender systems using matrix factorization, the ratings are arranged into a matrix R. The columns of R represent the users where its rows represent the items. The value of each not empty cell c_{ui} of R, corresponding to user u and item i, is a pair of values (r_{ui}, t_{ui}). r_{ui} is the rating given by u for the item i at time t_{ui}. An empty, i.e. missing, cell c_{ui} in R indicates that user u has not yet rated item i. Hence, the task of recommender system is to predict these missing rating values.

In its basic form (Basic MF), matrix factorization techniques try to capture the interactions between users and items that produce the different rating values. They try to approximate the matrix R of existing ratings as a product of two matrices:

$$R = P.Q \tag{2}$$

P and Q are called matrices of factors since they contain vectors of factors for the profiling of the users and the items, respectively. These matrices of factors are much more smaller than R. Thus, we gain in dimension reduction while getting predictive ratings simply by the following formula

$$\hat{r}_{ui} = p_u.q_i^T \tag{3}$$

where p_u and q_i are the vectors of factors, respectively in P and Q, corresponding to user u and item i.

In practice, it is almost not possible to obtain exactly R with the product of P and Q. Usually, some residuals remain. These latter constitute the error of prediction, i.e. its inaccuracy, which can be represented by a matrix E of errors having the same size than R. So, the previous equation can be changed to

$$R = P.Q + E \tag{4}$$

We can see that more the matrix E is close to a zero matrix, and more accurate will be the prediction. The process of training looks for the better values of P and Q such that the matrix E is the closest possible to a zero matrix. Thus, it tries to adjust all the values e_{ui} of the matrix E to zero using an expectation-maximization (EM) algorithm [10]. The EM algorithm computes a local minimum wherein the total sum of error values is one of the lowest according to initial ratings. In other words, it tries to minimize as good as possible the sum of quadratic errors $\sum_{ui} e_{ui}^2$ between the predictive ratings $\hat{r}_{ui}$ and the real ones r_{ui}. Errors are squared in order to avoid the effects of negative values in the sum, and minimizing $\sum_{ui} e_{ui}^2$ amounts to minimize each e_{ui}^2.

We have $e_{ui} \stackrel{def}{=} r_{ui} - \hat{r}_{ui}$. By using the vectors of factors p_u and q_i, we obtain that $e_{ui} \stackrel{def}{=} r_{ui} - p_u.q_i^T$. If we denote by K the number of considered factors, we have

$$\sum_{ui} e_{ui}^2 = \sum_{ui} (r_{ui} - p_u.q_i^T)^2 = \sum_{ui} (r_{ui} - \sum_{k}^{K} p_{uk}.q_{ki})^2 \tag{5}$$

To minimize the quadratic errors, we compute the differential (i.e., the gradients) of the above formula 5 to determine the part of change due to each factor (p_{uk} and q_{ki}):

$$\frac{\partial e_{ui}^2}{\partial p_{uk}} = -2.e_{ui}.q_{ki} \ , \qquad \frac{\partial e_{ui}^2}{\partial q_{ki}} = -2.e_{ui}.p_{uk} \tag{6}$$

We update p_{uk} and q_{ki} in the opposite direction of the gradients in order to decrease the errors and thus obtain a better approximation of the real ratings.

$$p_{uk} \leftarrow p_{uk} + \lambda.(2.e_{ui}.q_{ki} - \beta.p_{uk}) \tag{7}$$

$$q_{ki} \leftarrow q_{ki} + \lambda.(2.e_{ui}.p_{uk} - \beta.q_{ki}) \tag{8}$$

Here λ is the learning rate used to avoid overfitting and β is a regularization factor which serves to prevent large values of p_{uk} and q_{ki}. The EM algorithm iterates on equations 5, 7 and 8 until the sum of the quadratic errors does not decrease any more. This process corresponds to the training step.

After this training, the predictions $\hat{r}_{ui}$ are computed through the products $p_u.q_i^T$ of both vectors of factors. A sorting step allows to find the most relevant items to recommend to each user, i.e. the items with the greatest product values.

2.3 Biased MF

Several improvements to the above matrix factorization technique are proposed in the literature. One of these assumes that much of the observed variations in the rating values

is due to some effects associated with either the users or the items, independently of any interactions [29, 17, 23]. Indeed, there are always some users who tend to give higher (or lower) ratings than others, and some items may be higher (or lower) rated than others, because they are widely perceived as better (or worse) than the others. Basic MF can not capture these tendencies, thus some biases are introduced to highlight this rating variations. We call such techniques *Biased MF*. The biases reflect users or items tendencies. A first-order approximation of the biases involved in rating r_{ui} is as follows:

$$b_{ui} = \mu + b_u + b_i \tag{9}$$

b_{ui} is the global effect of the considered biases, it takes into account users tendencies and items perceptions. μ denotes the overall average rating (for all the items, by all the users). b_u and b_i indicate the observed deviations of user u, respectively item i, from the average. Hence, the equation 3 becomes

$$\hat{r}_{ui} = p_u.q_i^T + \mu + b_u + b_i \tag{10}$$

Since biases tend to capture much of the observed variations and can bring significant improvements, we consider that their accurate modeling is crucial [23, 16]. As for the factors p_{uk} and q_{ki} (equations 7 and 8), the biases have to be refined through a training step using the following equations:

$$b_i \leftarrow b_i + \lambda.(2.e_{ui} - \gamma.(b_u + b_i)) \tag{11}$$

$$b_u \leftarrow b_u + \lambda.(2.e_{ui} - \gamma.(b_u + b_i)) \tag{12}$$

where γ is another regularization factor. It plays the same role than β in equations 7 and 8.

2.4 Dynamicity and performance requirements

Dynamicity problem As told above, once a model is carried out, it remains static unless a new MF is computed. In real-world context, where new ratings happen continuously, the user interests evolve dynamically. Thus, the accuracy of the predictions decreases gradually and the computed profiles become obsolete after some time, since they do not take into account the new additions of ratings. To face this problem, recommender systems must regularly recompute their models, which represents an expensive task in terms of computation time. Hence, the dynamicity problem can be defined as follows: how to integrate the new ratings in the predictions without recomputing the model ? The goal is to maintain the accuracy of the predictions at a good level and postpone as far as possible the recomputation of the model.

We present in the following some important performance aspects, attempting to identify the requirements that the solutions for the dynamicity problem must satisfy.

Recommendation quality Assuming some fixed sets of users and items. We consider users continuously asking for items, and rating them. More precisely, a user asks for a short list of items with high probability of interest (i.e. high predicted rating), then selects and rates some of them, and so on. In such online recommendation scenario, the user expects the recommended items to be of high interest. We measure the quality of service in terms of the Mean Absolute Error (MAE) between the predicted and the real

ratings. We express the user requirement for quality, as a constraint on the MAE, which value must be greater than a given threshold ϵ.

$$MAE < \epsilon \tag{13}$$

Response time Another requirement for online recommendation is the response time tolerated by the end users. When a user asks for a recommendation, he expects to receive it almost immediately. Such requirement for online user demand is usually described by an upper bound along with a ratio of appliance [30]: 90% of the demands must be served in less than 5 seconds. This response time constraint forces us (1) to generate the model in advance, in order to anticipate the future demands, and (2) to limit the computational cost needed for the integration of the ratings arrived after the model generation.

Limited resource computation The last requirement concerns the overall computation cost of the recommendation. The solution must limit computations as much as possible. For instance, we could think of the model regeneration each time a new rating arrives, provided that we have unlimited computation resource. This will lead to a solution that meets the above requirements but that is not scalable. Therefore, we focus on solutions that take into account resource consumption in order to reduce it.

We can summarize the performance requirements into the following challenge: design a recommendation solution which provides sufficient quality, when generating the "top-quality" model takes a long time, when the model quality is decreasing over time, featuring fast recommendation delivery on user demand, and reducing the overall computation cost. In the following, we detail our method to tackle this challenge:

1. Combine clustering, MF and bias adjustment, in order to startup with a high quality model.

2. Continuous update of the biases in order to maintain satisfactory quality longer.

3 Online integration of new ratings

In this section, we present our regularized cluster-based matrix factorization. We start from the fact that many users tend to underestimate (or overestimate) the items they rate. A user exposes a tendency to rate above (or beyond) the average. We aim to quantify such tendency. A simple way to take this into account is to assign a single bias per user (see section 2.3). However, we observed that user tendency is not fixed: it depends on the items they rate. For some groups of items, the users tend to rate close to the average. While for some other groups (e.g., the items he really likes/dislikes), the users fail to rate objectively, either using extreme ratings, or keeping moderated ratings. To take into account this discrepancy, we define several biases per user, instead of a single one. We assign one bias per user per set of similar items, relying on existing clustering techniques to group similar items together. We expect that handling finer-grained biases will lead to more accurate recommendation. Another expected outcome is a lower update cost while integrating the new ratings, because the update implies only the items inside one group. In our context, the only information we known about the items is their ratings. Having additional information such as the item category is not the focus of this paper. Thus, we carry out groups of similar items through a clustering process that classifies the

items according to their ratings. We use a K-means algorithm for the clustering phase. Once the clusters are built, we assign a vector of biases to each user. Then, we apply a cluster-based MF on the ratings to generate the recommendation model. We now describe in more details the architecture of our solution, and the steps of the model generation process.

3.1 Architecture for online rating management

We detail the three layers of our recommendation system in Figure 1. First, the storage layer (lower layer) is responsible for storing the ratings and the generated model durably and efficiently. It manages two databases having different access patterns.

- The *database of ratings* assigned by users. It requires fast insert of new ratings (append only data structure) and point queries to retreive all the ratings of a given user, for a given group.
- The *database of factors and biases* generated by the factorization. It requires fast update of biases, and fast access to the factors and the biases of a given user.

Second, the recommendation processing layer (middle layer) has the following functions:

- *Offline factorization and clustering*: it generates the model, i.e., computes the factors in P and Q, and the biases of users. Then, it stores the model in the *database of factors and biases*. It reads the ratings in the *database of ratings*. It also runs the clustering procedure.
- an *items recommendation*: it provides recommendations upon user request. It queries the database of factors and biases to estimate the $\hat{r}_{ui}$ values. Then, it sorts them to keep the most relevant items to recommand. The item recommendation runs either on user demand, or precomputes recommandation in advance, for better response time.
- a *new ratings manager*: it handles the online integration of new ratings. Upon the arrival of a new rating, the online integration handles dynamic bias update. The new rating is also stored for subsequent model generation.

Third, the recommendation requests and the new ratings submissions (upper layer) represent the system workload.

We devise this architecture to better decouple the three main patterns that create (factorization), update (online integration) an read (item recommendation) the model respectively. This allows to control and postpone the factorization, depending on the quality of the item recommendation.

3.2 K-means clustering

The purpose of the K-means clustering algorithm (denoted KCA here after) is to divide p-dimensional items into K clusters C_k such that intra-cluster variance is the lowest possible. In other words, we minimize the next function

$$\theta_K = \sum_{k=1}^{K} \sum_{i \in C_k} dist(r_i, c_k) \tag{14}$$

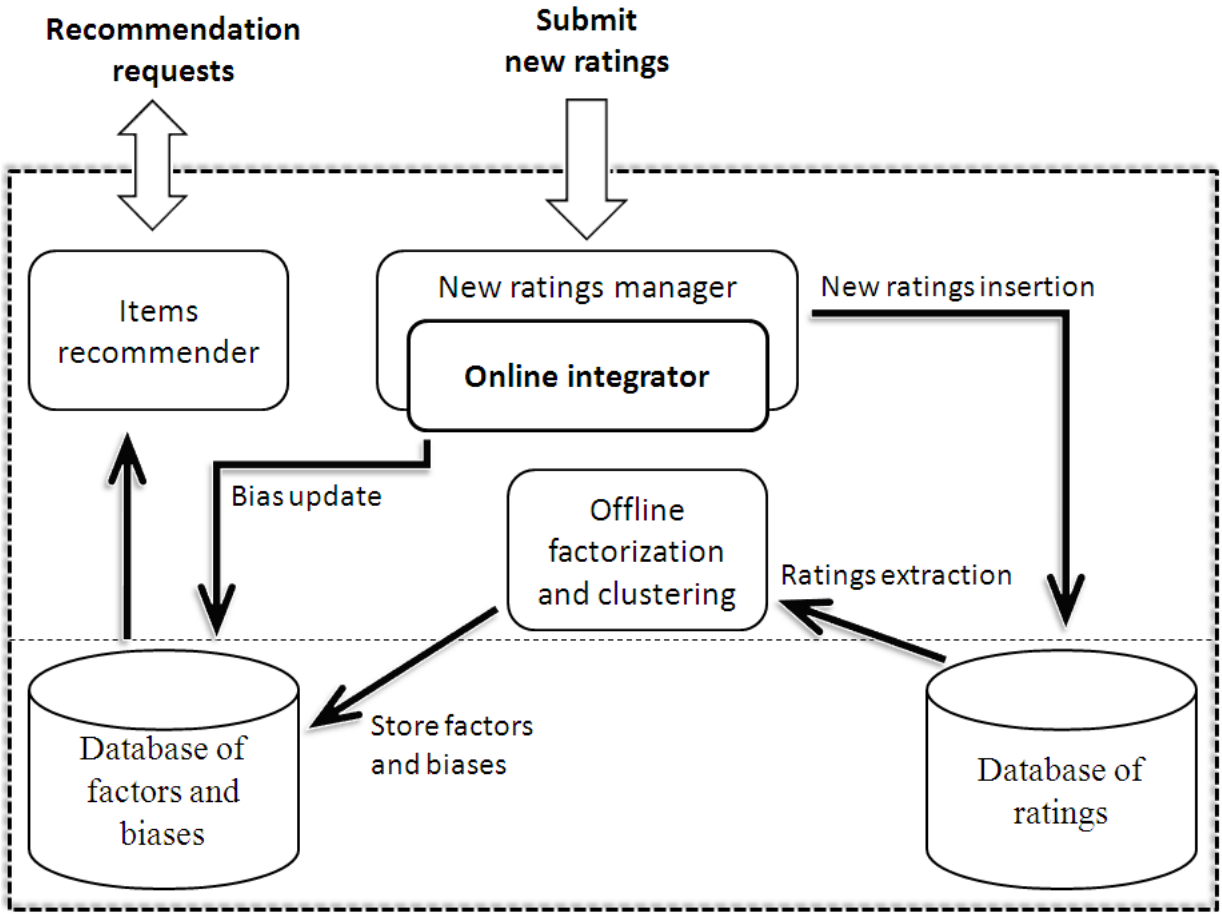

Figure 1: Architecture of our recommender system

Where *dist* is a metric of distance between two items and c_k the centroid, i.e. the mean item in C_k. The squared Euclidean distance is generally used as metric of distance.

$$dist(r_i, c_k) = \sum_{j \in J_i} (r_{ij} - c_{kj})^2 \tag{15}$$

In this formula, J_i represents the set of defined dimensions for item r_i, which corresponds in our case to the total number of users having rated the item r_i.

Different implementations of the KCA can be found in the literature [20, 13, 19]. The one proposed by Hartigan and Wong (see Algorithm 1) is well know as having a particular appeal because of its simplicity and computational speed. We choose this algorithm as the basis of our implementation.

3.3 Initializing the K-means clustering

One identified shortcoming of the K-means clustering is that it is very influenced by the starting centroids. Indeed, the K-means does not ensure that we have the best clustering but permits to get only one of the best possibilities. The choice of the starting centroids determines the quality of the clustering. Thus, it is very important to choose them. This issue is known as the K-means clustering initialization. Different methods are proposed in the literature [7, 22, 4]. We used the one proposed in [22] in our implementation,

Algorithm 1 K-means clustering algorithm

Require: $K \in \mathbb{N}$, $\mathbf{R} = \{r_i \in \mathbb{N}^p\}$

1: Take randomly an initial set of K centroids $c_1^{(0)}$, $c_2^{(0)}$,... $c_k^{(0)}$
2: **repeat**
3: Let $C_k^{(t)}$ be the state of cluster C_k at the time t and $c_k^{(t)}$ its cendroid at this time.
4: **Assignment step:** assign each observation in $\mathbf{R}$ to the cluster with the closest centroid
5: $C_k^{(t)} = \left\{ r_i : dist(r_i, c_k^{(t)}) \leq dist(r_i, c_l^{(t)}),\ l = 1, 2, ...K \right\}$
6: **Update step:** calculate the new mean to be the centroid of the observations in the cluster.
7: $c_k^{(t+1)} = \frac{1}{|C_k^{(t)}|} \sum_{r_i \in C_k^{(t)}} r_i$
8: **until** Assignments no longer change
9: **return** $C_1, C_2, ...C_K$

because it presents good performances in terms of minimizing the intra-cluster variance and does not seem to be statistically influenced by data dimension [21]. It attempts to get the initial centroids from observed data so that they are well separated from each other. Algorithm 2 describes this initialization method.

Algorithm 2 Initializing K-means clustering algorithm

Require: $K \in \mathbb{N}$, $\mathbf{R} = \{r_i \in \mathbb{N}^p\}$

1: Find all distances $dist(r_i, r_j)$ between each pair of items in the observed data $\mathbf{R}$.
2: Denote c_1 and c_2 as the items with the largest distance.
3: **repeat**
4: Let K^* be the number of current centroids.
5: For all $r_i \notin \{c_k, k = 1, 2, ...K^*\}$, compute $dist(r_i, c_k)$ for $k = 1, 2, ...K^*$. Keep the minimum $dist_{min}(r_i) = \min \{dist(r_i, c_k), k = 1, 2, ...K^*\}$
6: Set centroid $c_{K^*+1} = \max dist_{min}(r_i)$
7: **until** we have reached the number K of wanted centroids
8: **return** $c_1, c_2, ...c_K$

3.4 Cluster-based MF

After having carried out the groups of items C through clustering, i.e. the clusters, a vector of biases is allocated to each user. One bias for each group of items. Thus, we come down to observe local/group ratings variation in place of a single global ratings variation as used in previous approaches [23, 17, 28]. Our formula of prediction is as follows:

$$\hat{r}_{ui} = p_u . q_i^T + \mu_{C_{G(i)}} + b_{u,C_{G(i)}} + b_i \tag{16}$$

where $G(i)$ is a function which returns for any item i its group identifier. We denote by $\mu_{C_{G(i)}}$ the average ratings in $C_{G(i)}$ and by $b_{u,C_{G(i)}}$ the bias of user u for the group of items $C_{G(i)}$. We call $b_{u,C_{G(i)}}$ the *local user's bias*, i.e. the observed local deviation of user u. b_i represents the observed deviation of item i.

The bias $b_{u,C_{G(i)}}$ of user u for the cluster $G(i)$, to which the item i belongs, is derived from the existing ratings of item i in the group $C_{G(i)}$. For each rated item j in the cluster

$C_{G(i)}$, we calculate the deviation of user u for this item by $r_{uj} - \mu_{C_{G(i)}}$. The local bias of the user is obtained by taking its average deviation:

$$b_{u,C_{G(i)}} = \frac{1}{|C_{G(i)}|} \sum_{j \in C_{G(i)}} r_{uj} - \mu_{C_{G(i)}} \quad \forall j \in C_{G(i)}, s.t.\ r_{uj} > 0 \tag{17}$$

The algorithm 3 details the steps of our model generation process. Line 1 corresponds to the clustering phase. It is done by applying the algorithms 2 and 1 given before. Line 2 computes the bias of each item and the vector of biases of each user (as shown in equation 17). Lines 4 to 11 correspond to the most important part of the learning process. Lines 7, 8 and 9 update respectively the vectors of factors, the biases of the items and the biases of the users, using equations 7, 8, 11, and 12. They consider the error calculated in Line 6. Line 11 measures the global error. These steps are repeated until the global error given in equation 5 does not decrease any more or a certain number of iterations is met.

Algorithm 3 Cluster-based MF algorithm

Require: K: the number of wanted clusters, $\mathbf{R}$: the matrix $\mathbb{N}^{m*n}$ of ratings, k: the number of factors to consider, λ, β and γ

1: Compute the clusters $C_1, C_2, ...C_K$ from the observed data $\mathbf{R}$
2: For each item i and each user u, calculate the bias b_i of item i and the vector of biases $b_u = \left\{b_{u,C_{G(i)}}, G(i) \in 1, 2, ...K\right\}$
3: Initialize randomly, or with fixed values, the two matrices P and Q, respectively of dimensions $m * k$ and $k * n$.
4: **repeat**
5: **for all** $r_{ui} > 0 \in \mathbf{R}$ **do**
6: Compute e_{ui}
7: Update $p_{uk} \in P$, $q_{ki} \in Q$
8: Update b_i
9: Update also $b_{u,C_{G(i)}}$
10: **end for**
11: Calculate the global error $\sum_{r_{ui}>0 \in \mathbf{R}} e_{ui}^2$
12: **until** terminal condition is met
13: **return** P, Q, $\mu_G = \left\{\mu_{C_{G(i)}}, G(i) \in 1, 2, ...K\right\}$, b_i, b_u

After the generation of the model, the integration of additional ratings (those obtained after the MF) is obtained by adjusting local user bias $b_{u,C_{G(i)}}$. This is done by the execution of the instructions 4 to 12 in algorithm 3, except the lines 7 and 8 to keep unchanged the initial vectors of factors in P and Q.

4 Evaluation of the proposed technique

4.1 Complexity analysis

Let us denote by V the set of (existing) ratings in R.

$$V = \{r_{ui} \in R / r_{ui} > 0\} \tag{18}$$

The cost of our cluster-based matrix factorization solution (Algorithm 3) can be separated in two parts: the cost of matrix factorization and the cost of the clustering step. The

time complexity of the training of the whole model (matrix factorization) is $O(|V|.k.t)$, where k is the number of factors and t the maximum number of iterations (see Section 3.4). The clustering step is done using the algorithms 1 and 2. Its time complexity is $O(n.N_c.t)$ for the K-means clustering (Algorithm 1), and $O(\frac{n}{2}(N_c + n))$ for the initialization of the k-means (Alrorithm 2). n is the number of items to cluster, and N_c the number of clusters.

The online integration of new ratings is done in our approach simply by updating user biases. When a user u_1 rates by $r_{u_1 i}$ an item i, we update the bias of u_1 for only the group/cluster to which the item i belongs. Hence, we define the ratings of user u_1 as

$$V(u,.) = \{r_{uj} \in V / u = u_1\} \tag{19}$$

and all its ratings in the cluster $G(i)$ as

$$V(u, G(i)) = \left\{r_{uj} \in V(u,.) / j \in C_{G(i)}\right\} \tag{20}$$

Thus the time complexity of the integration of a new rating r_{ui} is $O(|V(u, G(i))|.t)$. Note that in the worst case this cost is equal to $O(|V(u,.)|.t)$, when all the ratings of the considered user belong to the same cluster. Let us stress that $V(u,.)$ is usually small. For instance, for Netflix the average size of $V(u,.)$ is 200 [1].The more the user ratings are distributed in different clusters, the more the cost of updating the user bias is small.

4.2 Experimental evaluation

In Section 2.4 we proposed to enhance the widely used MF model, coupling it with two techniques that tend to improve the quality of predictions: the preliminary clustering of the ratings before factorization, and the final adjustment of the predicted ratings using biases. This section presents the experiments we settled, in order to validate our approach. We remind that our approach consists of generating a high quality recommendation model based on incoming ratings. Then, we use that model for recommending items, as long as possible (provided that quality remains sufficient) up to next generated model is ready, and so on. Thus, the quality of our approach depends on two factors (i) the initial quality of the generated model, and (ii) the quality mitigation over time. Accordingly, we validate each factor independently, proceeding in two separated steps. Step 1 focuses on the initial quality of the model that has just been generated. Step 2 focuses on the quality mitigation, of our approach, over time.

Step 1: Validation of the initial quality We plan to show that our model yields good initial predictions compared to other commonly used models. We setup a fully informed environment, meaning that the model is aware of all the ratings that precede the prediction. This environment is optimal since it provides the maximal input to the model generation. Although this environment is rarely met in practice (it implies that no new ratings have occurred during the model generation), it ensures the most favorable conditions for every model. Thus it allows us for comparing several models when they expose their best strength. Our objective is to quantify the quality of our model that combines factorization with clustering and bias adjustment. To this end; we compare the accuracy of our model with two commonly used models: (i) the MF alone, and (ii) the biased MF (see Section 2.3). Note that, we do not compare our solution with the case of MF preceded by clustering without bias adjustment, since clustering does not improve the accuracy directly in its own. Actually, clustering allows for finer biases (one bias per cluster), which in turns yields better accuracy.

Step 2: Validation of the quality mitigation over time In the second validation step, we will check that the accuracy of prediction is actually decreasing over time. This aims to justify the relevance of our investigation to provide predictions which accuracy lasts longer. Then, we will measure the benefits of our approach (continuous bias update, based on new ratings) for keeping up the accuracy of prediction longer than others. In other words, our solution should expose a smaller quality decrease (i.e. a flatter slope) than other solutions. In consequence, it will imply less frequent model re-regeneration, saving a lot of computation work.

4.2.1 Implementation and Experimental setup

We implemented our proposition in python language (approximatively 4000 lines of code). Python is a powerful dynamic programming language that is well suited for rapid prototype developement Its standard library is large and very comprehensive. We used the *NumPy* package [2] for matrix manipulation. We ran our experiments on a linux computer (Intel/Xeon x 8 threads, 2.66 Ghz, 16 GB).

4.2.2 Datasets

The experiments use the publicly available MovieLens data sets [3]. These datasets are widely used by the recommendation system community [27]. They contain real ratings about movies, collected from MovieLens recommendation website since September 97. We use the dataset which contains 100K ratings for 1682 movies (i.e. items) by 943 users.

A rating is an integer in [1-5]. Each user has rated at least 20 movies. We made preliminary tests to calibrate the four parameters of the model: $\lambda = 0.001$, $\beta = 0.02$, $\gamma = 0.025$ and the number $K = 2$ of clusters. The λ, β, and γ values are close to the ones suggested in [23].

4.2.3 Experiment 1: Initial quality

The objective of the experiment 1 is to measure the initial quality of the model. We check the intuitive rule stating that the more ratings we take as input, the more quality we get. We split the dataset at date d_0: the training set ends at d_0, the test set starts at d_0. We choose d_0 such that the test set represents the last 20% of the dataset. We are varying the training set cardinality from 10K to 80K ratings by varying the start date of the training set. More precisely, for the i^{th} run, we select the ratings in $[d_i, d_0]$, choosing d_i such that the training set contains (10000*i) ratings. Note that every training set ends at d_0, this ensures we always use the most recent ratings to generate the model at date d_0. We take the quality of the i^{th} run as the ratio of the calculated MAE to the best one obtained from all. Let us note that this best MAE (i.e. smallest MAE) is given by our solution and its value equals 0.706.

Figure 2 shows that the quality is increasing almost linearly up to 50K ratings, this confirms our intuitions. Beyond that point, adding more rating into the training set does not bring much quality improvement as before: old ratings appear to be less relevant, this is possibly due to mind-changing users. On this figure, we also plot the respective quality of the models that we are comparing with: basic MF and biased MF. Hopefully, our model always make progress better than the two other models, thanks to finer-grain cluster based bias adjustment. Although our model begins with the lowest quality, it ends with the best quality. Indeed, with the first training sets we have'nt enough data to

make goods clusters. Combined to the fact that the users haven't yet rated a lot of items which harms the adjustment of biases, we expected these cases. But from 50K ratings as training set, we remark an improvement with our model. It is not only our model which is affected by the smallness of training set, the biased MF suffers from that also. This explains why the basic MF presents the best quality for the two smallest training sets (under 30K ratings). Let us note that our model has the highest relative progression in quality with an average of 3.92% when the biased MF has 3.42% and the basic MF 1.74% benefit.

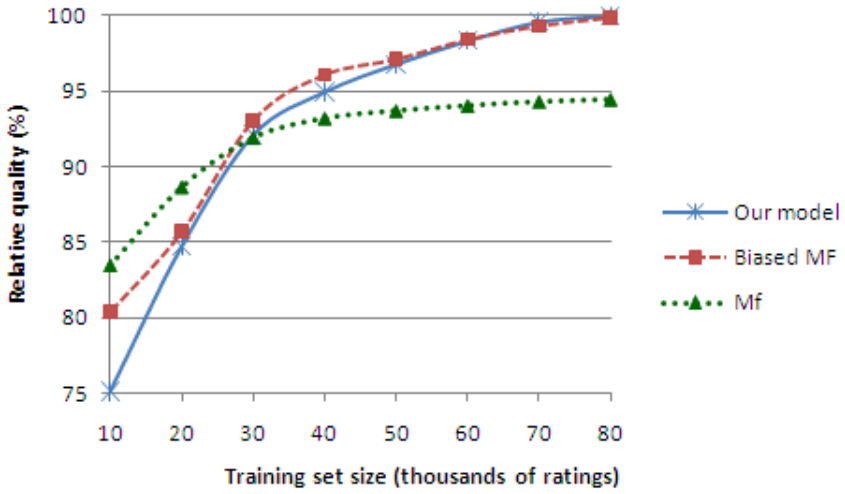

Figure 2: Quality increases with the amount of ratings

4.2.4 Experiment 2: Quality over time

The objective is to show that our model is robust to time, i.e., it still yields rather good quality predictions even when the model becomes out-of-date. We define the test set and the training set as follows. The test set remains identical; it contains all the ratings since date d_0, up to the end of the dataset. The training set size is fixed to 30K ratings, because in the previous experiment, the positive effect of adding biases (i.e. the biased MF over-performs the basic MF) appeared with a training set containing that number of ratings. Then, we define several training sets, such that the model becomes increasingly out-of-date. To this end, we are varying the end date (named d_{end}) of the training sets, in order to increase the gap between the model generation (at d_{end}) and the model use (starting at d_0). More precisely, for each training set, we set the number of missed ratings (named M) that must occur during $[d_{end}, d_0]$. Then, according to M, we set the start/end date of each training set. On Figure 3, M is varying from 0 to 50K, along the x-axis. We report, on the y-axis the quality expressed as a ratio relative to the minimal MAE, MAE_{min}, obtained for $M = 0$ (MAE_{min} equals 0.791 for our model, and 0.781 for the biased MF). We see that the quality of the biased MF is decreasing over time as we expected. For our model, the quality also follows a decreasing tendency, but with some fluctuations. Such oscillations in quality are due to the non-uniform distribution of the ratings for each item, that is quite accentuated by our training set size. We expect that our cluster-based approach better captures this non-uniformity for larger training sets.

Furthermore, we also show the importance of taking into account the new ratings.

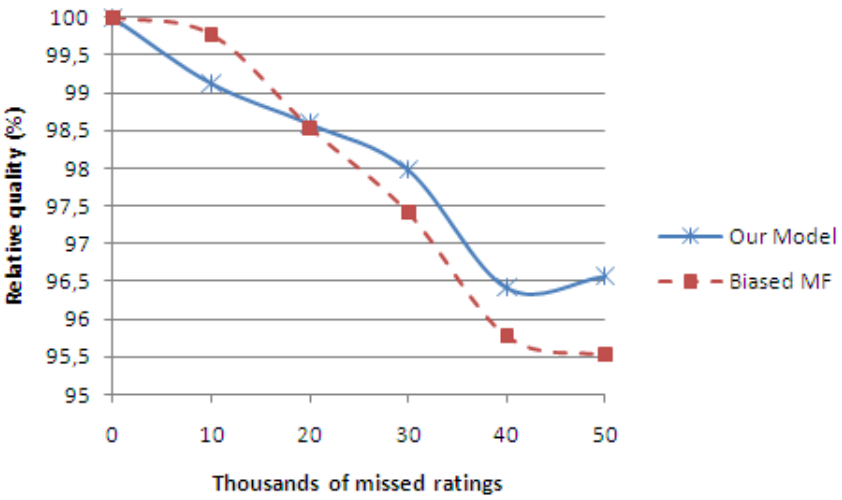

Figure 3: Quality over time

Using the same training sets, we now adjust the users' biases with the missed ratings (occuring in $[d_{end}, d_0]$), as said in Section 3.4. Accordingly, we also consider the missed ratings to adjust the single bias of the biased MF solution we are comparing with. We check that such adjustement is fast enough (from milliseconds to few seconds) to add neglectible overhead on the online recommandation task. On Figure 4, we report the relative quality improvement, in comparison with Figure 3, for the same x values. We see that our model performs up to twice better than biased MF. On average, we get 2.42% of quality improvement, while biased MF yields only 1.47% improvement. This confirms the benefit and the feasability of online integration of new ratings.

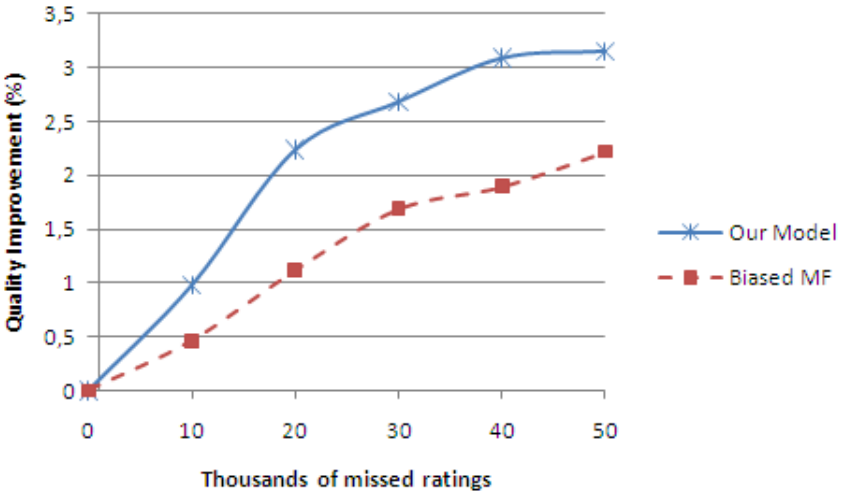

Figure 4: Quality's improvement by new ratings

4.2.5 Towards tuning the training set size

In this section, we generalize from the results of Sections 4.2.3 and , and explain the dependency between the training set size (TS) and the number of missed ratings (M). Then, we investigate the combined influence of both TS and M on the quality, in the case of intensive new ratings arrival. Finally, we devise a method to assess the optimal TS. Experiment 1 showed that taking into account more rating in the training set, improves the recommendation quality. However, to highlight the effect of the training set size on the quality, we assumed that no ratings arrived during the factorization. In other words, we assumed that the factorization time is smaller than the rating inter-arrival time. We now relax this assumption to better reflect the general case, and we quantify the factorization time $Tfact$. It depends on the training set size as follows: $Tfact = f(TS)$ where f is monotonic and increasing. In our context, we observed that $Tfact$ is actually longer than the inter-arrival time, thus several ratings arrived during the factorization, before the recommender is ready for use. We get: $M = Tfact * Th$ where Th is the input throughput. In experiment 2, we showed that a large M tends to decrease the quality of a given training set. Intuitively, the impact of M on the quality depends on how much the test set relies on recent ratings rather than on older ratings. For instance, most users rate movies soon after their release date. In consequence, increasing the training set size is influencing the resulting quality on two opposite directions since TS and M are increasing all together. On one side, a small TS does not rely on enough ratings to make factorization relevant, thus it yields poor quality. On the other side, a large TS implies a large M, that makes factorization out-of-date, thus yields poor quality as well. In between, there exists an optimal TS value (called TS_{opt}) that yields optimal quality.

A practical method to assess TS_{opt} is to process a set of runs for several TS growing values. For each run, one fix the bounds of the training set such that it precedes the test set with a delay of M/Th. Then, plotting the meassured quality should highlight the TS_{opt} point. Depending on the context, the relative impact of the growing delay (M) vs. the growing TS might be low. In that case, which is that of MovieLens, instead of figuring out TS_{opt}, one fall back to figure out the minimal TS value that is close enough (wrt. the required quality of service) to the maximal quality obtained with the largest available TS.

5 Related work

Previous research on online integration of new ratings focused particularly on the "new user/item" problem, in order to let the recommender system able to propose predictions for new registered users and items [9, 25, 24].

In [9], the author proposes an incremental clustering solution to tackle the new user problem. He performs an initial K-medoids clustering with the PAM (Partitioning Around Medoids) algorithm. Then, for the integration of new users, he developed a cluster-updating algorithm capable of swapping the medoids and producing new clusters. Sarwar et al. [25] adapted the Latent Semantic Indexing (LSI) technique, applied in information retrieval (IR) to solve the problem of synonymy and polysemy. They used this technique to reduce the dimensionality of the user-item ratings matrix. The proposed *Folding-in* technique uses a projection method to fold the new users and items in the already computed model. This solution is very efficient in terms of computational time, but its quality have to be compared to similar techniques used in IR, like *SVD-updating* [6].

Rendle et al. [24] extend the "new user/item" problem to focus on users (and items) with a small rating profile. They propose an approximization method that updates the matrices of an existing model (previously generated by MF). The *UserUpdate* algorithm they propose retrains the factor vector for the concerned user and keeps all other entries in the matrix unchanged. Its time complexity is $O(|V(u,.)|.k.t)$, where k is the given number of factors and t the number of iterations. In contrast with our solution, the whole factor vector of the user is retrained (i.e. his rating profile for all the items), which makes their solution more time consuming than ours ($O(|V(u,G(i))|.t)$, see Section 4.1). They also not consider user biases, which might be very important for the accuracy of the predictions.

Cao et al. [8] point the problem of data dynamicity in latent factors detection approaches. They propose an *online* nonnegative matrix factorization (ONMF) algorithm that detects latent factors and tracks their evolution when the data evolve. Let us remind that a nonnegative matrix factorization is a factorization where all the factors in both matrices P and Q are positive. They base their solution on the *Full-Rang Decomposition Theorem*, which states that: for two full rank decompositions $P_1.Q_1$ and $P_2.Q_2$ of a matrix R, there exists one invertible matrix X satisfying $P_1 = X.P_2$ and $Q_1 = X^{-1}.Q_2$. They use this relation to integrate the new ratings. Although the process seems to be relatively fast, its computation time is greater than ours. This is due to the fact that their technique updates the whole profiles of all the users where our solution limits the computations to the bias of the concerned user.

6 Conclusion

In this paper, we have presented a cluster-based matrix factorization (MF) approach which aims to overcome a major drawback of existing MF solutions: their inability to deal with dynamicity in a real-world context. To this end, we have introduced a multi-biases model that leverages clustering to capture fine-grained users' profiles more accurately than the existing single bias approach. We have proposed a detailed algorithm to generate our model, then to dynamically incorporate new ratings without regenerating the whole model. We have implemented our solution, as well as two other related ones. We have performed extensive experiments on the widely used MovieLens datasets, in order to validate both quality and performance. Qualitative results place our solution close to its competitors, while offering better quality preservation over time. Performance results expose fast integration of new ratings; that makes our solution viable for online recommendation systems. Finally, we have devised a practical method to tune the training set size in case of intensive arrival of new ratings. This is a first step towards quality aware cost based optimization of online recommendation.

References

[1] http://en.wikipedia.org/wiki/netflix_prize.

[2] http://numpy.scipy.org/.

[3] http://www.grouplens.org/node/73.

[4] D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA

'07, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[5] R. M. Bell, J. Bennett, Y. Koren, and C. Volinsky. The million dollar programming prize. *IEEE Spectr.*, 46:28–33, May 2009.

[6] M. W. Berry, S. T. Dumais, and G. W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Rev.*, 37:573–595, December 1995.

[7] P. S. Bradley and U. M. Fayyad. Refining initial points for k-means clustering. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 91–99, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[8] B. Cao, D. Shen, J.-T. Sun, X. Wang, Q. Yang, and Z. Chen. Detect and track latent factors with online nonnegative matrix factorization. In *Proceedings of the 20th international joint conference on Artifical intelligence*, pages 2689–2694, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[9] P. Chakraborty. A scalable collaborative filtering based recommender system using incremental clustering. In *Advance Computing Conference, 2009. IACC 2009. IEEE International*, pages 1526 –1529, March 2009.

[10] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[11] M. B. Dias, D. Locher, M. Li, W. El-Deredy, and P. J. Lisboa. The value of personalised recommender systems to e-business: a case study. In *Proceedings of the 2008 ACM conference on Recommender systems*, RecSys '08, pages 291–294, New York, NY, USA, 2008. ACM.

[12] D. M. Fleder and K. Hosanagar. Recommender systems and their impact on sales diversity. In *Proceedings of the 8th ACM conference on Electronic commerce*, EC '07, pages 192–199, New York, NY, USA, 2007. ACM.

[13] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):pp. 100–108, 1979.

[14] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22:5–53, January 2004.

[15] D. Jannach and K. Hegelich. A case study on the effectiveness of recommendations in the mobile internet. In L. D. Bergman, A. Tuzhilin, R. D. Burke, A. Felfernig, and L. Schmidt-Thieme, editors, *RecSys*, pages 205–208. ACM, 2009.

[16] Y. Koren. The bellkor solution to the netflix grand prize, August 2009.

[17] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42:30–37, August 2009.

[18] G. Linden, B. Smith, and J. York. Industry report: Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Distributed Systems Online*, 4(1), 2003.

[19] S. P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28:129–137, 1982.

[20] J. B. Macqueen. Some methods of classification and analysis of multivariate observations. In L. M. Le Cam and J. J. Neyman, editors, *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, 1967.

[21] R. Maitra, A. D. Peterson, and A. P. Ghosh. A systematic evaluation of different methods for initializing the k-means clustering algorithm. 2010.

[22] B. Mirkin. *Clustering for data mining : a data recovery approach.* Chapman & Hall/CRC, Boca Raton, FL, 2005.

[23] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proc. KDD Cup Workshop at SIGKDD'07, 13th ACM Int. Conf. on Knowledge Discovery and Data Mining*, pages 39–42, 2007.

[24] S. Rendle and L. Schmidt-Thieme. Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In P. Pu, D. G. Bridge, B. Mobasher, and F. Ricci, editors, *RecSys*, pages 251–258. ACM, 2008.

[25] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Incremental singular value decomposition algorithms for highly scalable recommender systems. In *Proceedings of the 5th International Conference in Computers and Information Technology*, 2002.

[26] J. B. Schafer, J. Konstan, and J. Riedi. Recommender systems in e-commerce. In *Proceedings of the 1st ACM conference on Electronic commerce*, EC '99, pages 158–166, New York, NY, USA, 1999. ACM.

[27] X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:4:2–4:2, January 2009.

[28] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Investigation of various matrix factorization methods for large recommender systems. In *Proceedings of the 2nd KDD Workshop on Large-Scale Recommender Systems and the Netflix Prize Competition*, NETFLIX '08, pages 6:1–6:8, New York, NY, USA, 2008. ACM.

[29] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Scalable collaborative filtering approaches for large recommender systems. *J. Mach. Learn. Res.*, 10:623–656, June 2009.

[30] TPC-Council. Tpc benchmark c, rev 5.11. Technical report, Transaction Processing Performance Council, 2010.