Fast ALS-based Matrix Factorization for Explicit and Implicit Feedback Datasets

István Pilászy, Dávid Zibriczky
Gravity Research & Development
Expo tér 5.-7.
Budapest, Hungary
pila,david.zibriczky@gravityrd.com

Domonkos Tikk*
Institute for Computer Science
Humboldt-University of Berlin
Unter den Linden 6
Berlin, Germany
tikk@informatik.hu-berlin.de

ABSTRACT

Alternating least squares (ALS) is a powerful matrix factorization (MF) algorithm for both explicit and implicit feedback based recommender systems. As shown in many articles, increasing the number of latent factors (denoted by K) boosts the prediction accuracy of MF based recommender systems, including ALS as well. The price of the better accuracy is paid by the increased running time: the running time of the original version of ALS is proportional to K^3. Yet, the running time of model building can be important in recommendation systems; if the model cannot keep up with the changing item portfolio and/or user profile, the prediction accuracy can be degraded.

In this paper we present novel and fast ALS variants both for the implicit and explicit feedback datasets, which offers better trade-off between running time and accuracy. Due to the significantly lower computational complexity of the algorithm—linear in terms of K—the model being generated under the same amount of time is more accurate, since the faster training enables to build model with more latent factors. We demonstrate the efficiency of our ALS variants on two datasets using two performance measures, RMSE and average relative position (ARP), and show that either a significantly more accurate model can be generated under the same amount of time or a model with similar prediction accuracy can be created faster; for explicit feedback the speed-up factor can be even 5–10.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*parameter learning*

General Terms

Algorithms, Experimentation

Keywords

collaborative filtering, matrix factorization, alternating least squares, ridge regression, implicit and explicit feedback, computational complexity

*D. Tikk is currently on leave from Dept. of Telecom. and Media Informatics, Budapest University of Technology and Economics, Magyar Tudósok krt. 2., Budapest, Hungary

1. INTRODUCTION

Recommender systems suggest personalized recommendations on items to users based on various kinds of information on users and items. Recommender systems intend to model user preferences on items and aim to recommend such items the user will probably like. User preference information is twofold: explicit and implicit feedback. The former mainly includes opinion expression via *ratings* of items on a predefined scale, while the latter consists of other user activities, such as purchasing, viewing, renting or searching of items.

The commercial applications of recommender systems have the ultimate goal of profit maximization. However, due to business reasons revenue data are hardly available, therefore in the scientific context one may optimize recommender algorithms for such performance measures that supposedly correlate well with the business goals. For this purpose error measures are typically applied, such as the root mean square error (RMSE) or the mean absolute error (MAE). Another alternative is the ranking based average relative position (ARP, we define it later).

Most recommendation algorithms exhibit such parameter(s) that enable a complexity–accuracy trade-off. In case of latent factor models, e.g. matrix factorization based algorithms, the most important parameter of the model is the number of latent factors. For neighborhood-based models the number of neighbors considered affects the complexity of a model the most. As it was shown in numerous publications (see e.g. [7, 13, 12, 6]), the higher its complexity, the more accurate the model. On the other hand, the more complex the model, the higher its training time. Albeit the accuracy is a prominently important feature of recommendation systems, it is equally important that the model is computable under a reasonable amount of time; here the time constraint obviously depends on the application domain. Though, the two top-competitors of the Netflix Prize achieved 10.06 % improvement on the test set, Netflix enhanced only very few of algorithms and blending methods, those ones that are simple, accurate enough, and can scale up well to very large dataset[1].

The faster training time of an algorithm, allowing for various running time–accuracy trade-off scenarios, can be exploited in commercial applications in several ways. First, services with typically short item lifetime—such as news feeds, auction portals, non-repetitive cultural events—require frequent periodical retraining of the recommendation model otherwise the model becomes outdated and inaccurate. The incremental update of the recommendation model with interactions on new items offers only a temporal solution, because this technique does not handle item-item relations. Similarly, but in much less extent, the user profiles can also be eroded with time. A faster algorithm enables more frequent retraining and constantly higher recommendation quality at the same hardware cost. Second, faster training time enables to build a better recommendation model under the same amount of time. This can be either achieved by building a more accurate recommendation model with a larger parameter set (for example the number of latent factors can be increased at latent factor models), or running the same training setting for more epochs. Third, the saved computational overhead obtained by replacing a slower method can be devoted for other purposes.

This paper is organized as follows. Section 2 introduces alternating least squares as a matrix factorization based recommender algorithm for both explicit and implicit feedback datasets, describes the computational complexity of the naive implementation and the one using the Sherman–Morrison formula (SMF). Section 3 presents our fast ALS variants: an efficient approximate ridge regression implementation and its application for ALS based recommender algorithms. We report on and analyze the results obtained with our proposed algorithms in Section 4. A brief review of related work is presented in Section 5. We conclude the paper and give hints on future work in 6.

2. MATRIX FACTORIZATION

We use the following notation in the paper. N and M denote the number of users and items, resp. We use $u \in \{1, \ldots, N\}$ as index for users, and $i, j \in \{1, \ldots, M\}$ as indices for items. The rating of user u on item i is r_{ui}, and its prediction is $\hat{r}_{ui}$. All r_{ui} ratings are arranged in $\mathbf{R}$; $\mathcal{R}$ denotes the set of (u,i) indexes of $\mathbf{R}$ where (a) a rating is provided (for explicit feedback), (b) a positive feedback is provided (for implicit feedback). $\mathcal{R}^{\pm}$ denotes each (u,i) pair of $\mathbf{R}$, i.e. $|\mathcal{R}^{\pm}| = N \cdot M$.

Matrix factorization approaches have been applied successfully for both rating-based and implicit feedback-based collaborative filtering (CF) problems [2, 3, 7, 9, 11, 12]. MF methods perform a so-called low rank matrix approximation: the matrix $\mathbf{R}$ is approximated as a product of two lower rank matrices: $\mathbf{R} \approx \mathbf{P}\mathbf{Q}^{\mathrm{T}}$, where $\mathbf{P} \in \mathbb{R}^{N \times K}$ is the user feature matrix, $\mathbf{Q} \in \mathbb{R}^{M \times K}$ is the item feature matrix, K is the number of features that is a predefined constant, and the approximation is only performed at $(u,i) \in \mathcal{R}$ positions.

The r_{ui} element of $\mathbf{R}$ is approximated by

$$\hat{r}_{ui} = \mathbf{p}_u^{\mathrm{T}} \mathbf{q}_i.$$

[1]`http://www.newscientist.com/article/dn17823-photo-finish-for-the-1m-movie-prediction-prize.html`

Here $\mathbf{p}_u \in \mathbb{R}^{K \times 1}$ is the user feature vector, the u-th row of $\mathbf{P}$, and $\mathbf{q}_i \in \mathbb{R}^{K \times 1}$ is the item feature vector, the i-th row of $\mathbf{Q}$. The approximation aims to minimize the error of prediction, $e_{ui} = r_{ui} - \hat{r}_{ui}$, while keeping the Euclidean norm of the user and item feature vectors small:

$$(\mathbf{P}^*, \mathbf{Q}^*) = \operatorname*{arg\,min}_{\mathbf{P},\mathbf{Q}} \sum_{(u,i) \in \mathcal{R}} \left(e_{ui}^2 + \lambda \mathbf{p}_u^{\mathrm{T}} \mathbf{p}_u + \lambda \mathbf{q}_i^{\mathrm{T}} \mathbf{q}_i \right).$$

The predefined regularization parameter λ trades off between small training error and small model weights.

There are two commonly used approaches to perform the approximate matrix factorization task: gradient descent [11] and alternating least squares (ALS) [2]. In this paper we investigate ALS, which can be effectively used for also for implicit feedback datasets. ALS is based on ridge regression (RR), therefore we describe that first.

2.1 Ridge regression

Let us assume that we are given n training examples: $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^{K \times 1}$, forming the matrix $\mathbf{X} \in \mathbb{R}^{n \times K}$, with corresponding target values $y_1, \ldots, y_n \in \mathbb{R}$ that are arranged in the vector of target variables $\mathbf{y} \in \mathbb{R}^{n \times 1}$. With ridge regression one aims at finding a $\mathbf{w} \in \mathbb{R}^{K \times 1}$ vector such that both the error of approximating y_i with $\mathbf{w}^{\mathrm{T}} \mathbf{x}_i$ and the quantity $\mathbf{w}^{\mathrm{T}} \mathbf{w}$ are small. Ridge regression seeks the minimum of the cost function

$$\lambda \mathbf{w}^{\mathrm{T}} \mathbf{w} + \sum_{i=1}^{n} (\mathbf{w}^{\mathrm{T}} \mathbf{x}_i - y_i)^2, \tag{1}$$

where $\lambda > 0$ is the regularization factor. The cost function (1) can be easily minimized with

$$\mathbf{w} = (\lambda \mathbf{I} + \mathbf{A})^{-1} \mathbf{d}, \tag{2}$$

where $I_K \in \mathbb{R}^{K \times K}$ is the identity matrix, and

$$\mathbf{A} = \mathbf{X}^{\mathrm{T}} \mathbf{X}, \qquad \mathbf{d} = \mathbf{X}^{\mathrm{T}} \mathbf{y}. \tag{3}$$

The determination of (2) is dominated by the computation of $\mathbf{A}$, which is $O(K^2 n)$, and the matrix inversion, which is $O(K^3)$.

When some training examples are more important than others, we can assign the training examples non-negative importance weight (confidence level); denoted here by c_i. At this *weighted ridge regression* the goal is to minimize the cost function:

$$\lambda \mathbf{w}^{\mathrm{T}} \mathbf{w} + \sum_{i=1}^{n} c_i (\mathbf{w}^{\mathrm{T}} \mathbf{x}_i - y_i)^2. \tag{4}$$

Consequently, the computation of $\mathbf{A}$ and $\mathbf{d}$ changes to:

$$\mathbf{A} = \mathbf{X}^{\mathrm{T}} \mathbf{C} \mathbf{X} = \sum_{i=1}^{n} c_i \mathbf{x}_i \mathbf{x}_i^{\mathrm{T}}, \qquad \mathbf{d} = \mathbf{X}^{\mathrm{T}} \mathbf{C} \mathbf{y} = \sum_{i=1}^{n} c_i \mathbf{x}_i y_i, \tag{5}$$

where $\mathbf{C} \in \mathbb{R}^{n \times n}$ is a diagonal matrix containing the c_i values along its diagonal.

2.2 ALS

Bell and Koren [2] proposed the alternating least squares approach for CF, which is based on RR. ALS alternates between two steps: the $\mathbf{P}$-step fixes $\mathbf{Q}$ and recomputes $\mathbf{P}$, the $\mathbf{Q}$-step fixes $\mathbf{P}$ and recomputes $\mathbf{Q}$. The recomputation of $\mathbf{P}$ is performed by solving a separate RR problem for each user: for the u-th user it takes the feature vector ($\mathbf{q}_i$) of items rated by the user as input variables, and the value of the ratings (r_{ui}) as output variable, and finds the optimal $\mathbf{p}_u$ by RR. Though, in the original paper a non-negative RR is performed, in this paper we relax this restriction and we allow for negative values in the solution.

Formally, let the matrix $\mathbf{Q}[u] \in \mathbb{R}^{n_u \times K}$ denote the restriction of $\mathbf{Q}$ to the items rated by user u, where $n_u = |\{i : (u,i) \in \mathcal{R}\}|$, the vector $\mathbf{r}_u \in \mathbb{R}^{n_u \times 1}$ denote the ratings given by the u-th user to the corresponding items, and let

$$\mathbf{A}_u = \mathbf{Q}[u]^{\mathrm{T}}\mathbf{Q}[u] = \sum_{i:(u,i)\in\mathcal{R}} \mathbf{q}_i\mathbf{q}_i^{\mathrm{T}}, \qquad (6)$$

$$\mathbf{d}_u = \mathbf{Q}[u]^{\mathrm{T}}\mathbf{r}_u = \sum_{i:(u,i)\in\mathcal{R}} r_{ui} \cdot \mathbf{q}_i \qquad (7)$$

where $\mathbf{A}_u$ is the covariance matrix of input (considering user u in context) and $\mathbf{d}_u$ is the input-output covariance vector. Then RR recomputes $\mathbf{p}_u$ as

$$\mathbf{p}_u = \left(\lambda n_u \mathbf{I} + \mathbf{A}_u\right)^{-1}\mathbf{d}_u. \qquad (8)$$

In the $\mathbf{P}$-step a ridge regression is solved for each user, which is $O(\sum_{u=1}^{N}(K^2 n_u + K^3))$ time, that is $O(K^2|\mathcal{R}| + NK^3)$ as noted in [7, 12]. Similarly, the $\mathbf{Q}$-step requires $O(K^2|\mathcal{R}| + MK^3)$. According to [2, 7], the number of recomputations needed ranges between 10 and a "few tens". It has been pointed out that larger K yields more accurate predictions [7, 12].

2.3 IALS

In the case of the implicit feedback, each user rates each item either positively (user u viewed item i) or negatively (user u did not view item i). This is in contrast with the explicit case, where only a small subset of items are rated (but usually on a finer scale). Consequently, at implicit feedback the recommendation algorithm should handle a full rating matrix, in contrast to the explicit feedback case. Given that the sparsity of the explicit rating matrix is usually less than 1%, the difference in the size of the data is usually several orders of magnitude.

Hu et al. [7] proposed an elegant *ALS* variant for *implicit* feedback datasets, referred to as IALS. The key idea behind their approach is to assign a preference value r_{ui} and a confidence level c_{ui} to each element of $\mathcal{R}^{\pm}$. Preference typically takes values 0 and 1, indicating whether a user watched a particular item or not. The confidence level is related to how confident we are about user preferences. If she watched the item for long, or visited it two times, we can be more confident. Then the optimal model is defined as:

$$(\mathbf{P}^*, \mathbf{Q}^*) = \underset{\mathbf{P},\mathbf{Q}}{\arg\min} \sum_{(u,i)\in\mathcal{R}^{\pm}} c_{ui}\cdot e_{ui}^2 + \lambda \sum_u \mathbf{p}_u^{\mathrm{T}}\mathbf{p}_u + \lambda \sum_i \mathbf{q}_i^{\mathrm{T}}\mathbf{q}_i,$$

where $e_{ui} = r_{ui} - \mathbf{p}_u^{\mathrm{T}}\mathbf{q}_i$ is the prediction error. Note that $\mathcal{R}^{\pm}$ contains all (u,i) pairs regardless of whether user u has watched item i or not. The authors use one restriction, namely if u has not watched i, then $r_{ui} = r_0 = 0$ and $c_{ui} = c_0 = 1$, where r_0 and c_0 are predefined constants, typically set to $r_0 = 0$ and $c_0 = 1$. Similarly to the explicit feedback case, the authors used ALS to solve the problem, but with the following two differences:

- handling confidence levels: the covariance matrix $\mathbf{A}_u$ and covariance vector $\mathbf{d}_u$ contains now a confidence-weighted sum (the weighted RR case);
- the sums are taken over all items (or users, since all user rated all items, and vice versa).

The authors of [7] proposed an efficient solution to the second issue: when recomputing $\mathbf{P}$, they calculate an initial covariance matrix $\mathbf{A}_0$ and a vector $\mathbf{d}_0$ for a virtual user who watched no items:

$$\mathbf{A}_0 = \sum_i c_0 \cdot \mathbf{q}_i\mathbf{q}_i^{\mathrm{T}}, \quad \mathbf{d}_0 = \sum_i c_0 \cdot r_0 \cdot \mathbf{q}_i. \qquad (9)$$

For now, let $\mathcal{R} = \{(u,i) : r_{ui} \neq r_0 \vee c_{ui} \neq c_0\}$. For user u the calculation of covariance matrix is starting with $\mathbf{A}_0$ and then for each positive implicit rating the did-not-watch-assumption is replaced obtaining:

$$\mathbf{A}_u = \mathbf{A}_0 + \sum_{i:\ (u,i)\in\mathcal{R}^{\pm}} (-c_0 + c_{ui}) \cdot \mathbf{q}_i^{\mathrm{T}}\mathbf{q}_i,$$

$$\mathbf{d}_u = \mathbf{d}_0 + \sum_{i:\ (u,i)\in\mathcal{R}^{\pm}} (-c_0 \cdot r_0 + c_{ui} \cdot r_{ui}) \cdot \mathbf{q}_i.$$

Then $\mathbf{p}_u$ is computed by RR:

$$\mathbf{p}_u = (\lambda\mathbf{I} + \mathbf{A}_u)^{-1}\mathbf{d}_u. \qquad (10)$$

In [7] it is also pointed out that the time complexity of recomputing $\mathbf{P}$ and $\mathbf{Q}$ are $O(K^2|\mathcal{R}^{\pm}| + K^3N)$ and $O(K^2|\mathcal{R}^{\pm}| + K^3M)$, resp. It is also shown experimentally that larger K values yield better prediction performance.

2.4 Speed-up from K^3 to K^2

In [10], we proposed to apply the Sherman–Morrison formula (SMF) to speed up ALS, while retaining exactly the same result of the algorithm. The Sherman–Morrison formula allows to efficiently update the inverse of the covariance matrix when the covariance matrix is updated by an example. To recompute $\mathbf{P}$, the time complexity is reduced from $O(K^2|\mathcal{R}| + NK^3)$ to $O(K^2|\mathcal{R}|)$, for both explicit and implicit feedback datasets.

However, this theoretical speed-up does not turn into real speed-up in many cases. For example, consider the case when $|\mathcal{R}| = 10^8$, $N = 500\,000$, $M = 20\,000$, and assume for the SMF-variant merely a constant factor of 2 hidden in the O-notation. The running times for various K values are the followings:

K	$2K^2\lvert\mathcal{R}\rvert + (N+M)K^3$	$4K^2\lvert\mathcal{R}\rvert$
50	$5.7 \cdot 10^{11}$	$10 \cdot 10^{11}$
100	$2.5 \cdot 10^{12}$	$4 \cdot 10^{12}$
200	$1.2 \cdot 10^{13}$	$1.6 \cdot 10^{13}$
500	$11.5 \cdot 10^{13}$	$10 \cdot 10^{13}$
1000	$7.2 \cdot 10^{14}$	$4 \cdot 10^{14}$

This means that even with the very large $K = 1000$ only a speed-up of factor 2 can be achieved. The SMF-variant is advantageous only when the average number of ratings per user and per item are both small. In the above case, these numbers are 200 and 5000 resp.

In the next sections we propose novel ALS variants that have time complexity $O(K|\mathcal{R}|)$, i.e. it is *linear in the number of latent factors.* The speed-up can be attributed to the approximate solution applied for the matrix inversion. Our ALS variants does not reproduce the results of ALS; with the same parameter setting usually it achieves slightly inferior prediction accuracy (see Section 4.2). On the other hand, thanks to the speed-up our ALS variants trained with a somewhat larger K can easily catch up with the accuracy of the original ALS under smaller amount of time.

3. MAIN CONTRIBUTIONS

3.1 RR1: Efficient ridge regression

In this section we propose an approximate algorithm for optimizing (4). Let $e_i = y_i - \mathbf{w}^{\mathrm{T}}\mathbf{x}_i$ denote the error on the i-th example. The proposed algorithm works as follows:

Let denote the k-th element of $\mathbf{w}$ by w_k ($k = 1, \ldots, K$). Initialize $\mathbf{w}$ with $\mathbf{0}$. Find the optimal solution only for w_1, assuming the other elements of $\mathbf{w}$ are fixed. Then optimize w_2, assuming other coordinates (w_1, w_3, w_4, $\ldots$, w_K) are fixed, and so on, in each step optimize only one element of $\mathbf{w}$. The loop of the consecutive separate optimization of w_k-s are repeated until a termination condition is met.

The recomputation of w_k is a univariate ridge regression, while the recomputation of the entire $\mathbf{w}$ is a multivariate ridge regression. To optimize only w_k, we need to solve the following ridge regression problem:

$$\forall_{i=1}^{n} \qquad w_k x_{ik} \approx y_i - \sum_{l \neq k} w_l x_{il}, \tag{11}$$

Here x_{il} is the (i, l)-th element of $\mathbf{X}$. We refer to k as the active feature.

At implementation, we can set w_k to 0 just before we optimize it. With this modification $y_i - \sum_{l \neq k} w_l x_{il}$ in (11) becomes equal to $e_{ui} = y_i - \mathbf{w}^{\mathrm{T}}\mathbf{x}_i$. However, when we change w_k to w'_k, we can easily update e_i to e'_i:

$$\forall_{i=1}^{n} \qquad e'_i = e_i - (w'_k - w_k)x_i, \tag{12}$$

In other words: after the optimization of w_{k-1}, but before the optimization of w_k, $y_i - \sum_{l \neq k} w_l x_{il}$ can be efficiently recomputed for each i in total in $O(n)$ time.

Algorithm 1 describes formally our RR1 algorithm for the weighted ridge regression case.

```
   Input: $n$: number of examples,
   $K$: number of features, $\lambda$: regularization factor,
   $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^{K \times 1}$: training examples,
   $y_1, \ldots, y_n$: target variables, $c_1, \ldots, c_n$: weight of
   examples,
   $L$: number of cycles, $\mathbf{w}$: initial weight vector, or $\mathbf{0}$.
   Output: $\mathbf{w}$: the optimized weight vector
 1 $\forall_{i=1}^{n} : e_i \leftarrow y_i - \mathbf{w}^{\mathrm{T}}\mathbf{x}_i$
 2 for $L$ times do
 3     one cycle:
 4     for $k \leftarrow 1$ to $K$ do
 5         $\forall_{i=1}^{n} : e_i \leftarrow e_i + w_k x_k$
 6         $w_k \leftarrow 0$
 7         $a \leftarrow \sum_{i=1}^{n} c_i x_{ik} x_{ik}$
 8         $d \leftarrow \sum_{i=1}^{n} c_i x_{ik} e_i$
 9         $w_k \leftarrow d/(\lambda + a)$.
10         $\forall_{i=1}^{n} : e_i \leftarrow e_i - w_k x_k$
11     end
12 end
```

Algorithm 1: Algorithm of weighted RR1

A few notes to the algorithm: Line 1 initializes the inner variables. The outermost **for**-cycle repeatedly optimizes each element of $\mathbf{w}$. Lines 5–6 sets w_k to 0 and adjusts the e_i variables accordingly. Lines 7–8 computes the 1×1 variant of $\mathbf{A}$ and $\mathbf{d}$ respectively. Line 9 executes the matrix inversion that is merely a division in the 1×1 case. Line 9 adjusts the e_i variables, since w_k was changed.

Note that RR1 uses only the diagonal of $\mathbf{A} \in \mathbb{R}^{K \times K}$ that can be precomputed and does not change between cycles. The computational complexity of one cycle is $O(nK)$. Assuming the number of cycles is L, and we precompute the diagonal of $\mathbf{A}$, the total running time is $O(LnK)$. If L is small a significant speed-up is achieved compared to $O(K^2 n)$.

3.2 ALS1: efficient ALS for explicit feedback

Recall that ALS alternates between the $\mathbf{P}$-step and the $\mathbf{Q}$-step. The emphasis is on recomputation: when it recomputes $\mathbf{P}$, it completely ignores the result of its previous recomputation. When RR is replaced with RR1, our fast ridge regression variant, in the $\mathbf{P}$-step we continue the optimization of user features starting with the latest $\mathbf{P}$ instead of a complete recomputation. Thus, we may refer to this step as "further optimization" instead of "recomputation".

Formally, we propose the following algorithm (ALS1):

- initialize $\mathbf{P}$ and $\mathbf{Q}$
- repeat until a termination condition is met:
  - $\mathbf{P}$-step: optimize $\mathbf{P}$: for each user u:
    apply RR1 on $\mathbf{p}_u$ as both the input and output weight vector, running it only for 1 cycle.
  - $\mathbf{Q}$-step: optimize $\mathbf{Q}$: for each item i:
    apply RR1 on $\mathbf{q}_i$ as both the input and output weight vector, running it only for 1 cycle.

The running time of one epoch of ALS1 is very favorable, $O(K|\mathcal{R}|)$. Recall that in ALS one epoch requires $O(K^2|\mathcal{R}|)$ time.

3.3 IALS1: efficient ALS for implicit feedback

In this section we restrict our investigation to the $\mathbf{P}$-step; the $\mathbf{Q}$-step can be treated analogously. As mentioned in Section 2.3, at implicit feedback we have a full rating matrix. To overcome the largely increased quantity of data, in IALS a virtual user was introduced who rated all items negatively (i.e. she watched nothing) [7]. Here we show how RR can be replaced with RR1 to obtain a fast implicit ALS version, and how the favorable efficiency of RR1 can be carried over adapting the virtual user trick to the case.

Generating synthetic examples.

So, let we again introduce a virtual user, with the corresponding covariance matrix $\mathbf{A}_0$ and vector $\mathbf{d}_0$ as in (9). RR uses only the $\mathbf{A}_0$ matrix and the $\mathbf{d}_0$ vector. On the other hand, RR1 uses the diagonal of $\mathbf{A}_0$, the $\mathbf{d}_0$ vector, and it also needs all the original examples (that generate $\mathbf{A}_0$ and $\mathbf{d}_0$), which makes inappropriate to combine the advantageous properties of RR1 and IALS.

However, instead of the using all the M items in the calculation of $\mathbf{A}_0 \in \mathbb{R}^{K \times K}$ and $\mathbf{d}_0 \in \mathbb{R}^{K \times 1}$ (cf. eq. (9)), we can generate them using a much smaller *synthetic item subset*, containing only exactly K synthetic items.

Next we show, how we can generate those K synthetic items (linear combination of items) such that properly rating these items will exactly reproduce $\mathbf{A}_0$ and $\mathbf{d}_0$, i.e. the covariance matrix and the input-output covariance of the user with all negative ratings. In this way, from the viewpoint of RR or RR1, rating these K items will be equivalent to rating all the M items negatively.

Since $\mathbf{A}_0$ is a symmetric real matrix (and also positive semi-definite), its eigenvalue decomposition

$$\mathbf{A}_0 = \mathbf{S}\mathbf{\Lambda}\mathbf{S}^{\mathrm{T}} \tag{13}$$

exists, where $\mathbf{S} \in \mathbb{R}^{K\times K}$ is the orthogonal matrix of the eigenvectors ($\mathbf{S}^{\mathrm{T}}\mathbf{S} = \mathbf{S}\mathbf{S}^{\mathrm{T}} = \mathbf{I}$), and $\mathbf{\Lambda} \in \mathbb{R}^{K\times K}$ is a diagonal matrix, containing the non-negative eigenvalues.

Let $\mathbf{G} = \sqrt{\Lambda}\mathbf{S}^{\mathrm{T}}$, then $\mathbf{G}^{\mathrm{T}} = \mathbf{S}\sqrt{\Lambda}$. Let $\mathbf{g}_j \in \mathbb{R}^{K\times 1}$ denote the j-th column of $\mathbf{G}^{\mathrm{T}}$ (here $j = 1, \ldots, K$), that is the feature vector of the j-th aforementioned synthetic example. Note that if a user rates the $\mathbf{g}_j$ examples with confidence level $c_j = 1$, the resulting covariance matrix, $\mathbf{A}_0' := \sum_{j=1}^{K} c_j \mathbf{g}_j \mathbf{g}_j^{\mathrm{T}} = \mathbf{G}^{\mathrm{T}}\mathbf{G}$ will be equal to $\mathbf{A}_0$.

Next we specify the ratings r_j on these $\mathbf{g}_j$ examples, such that $\mathbf{d}_0$ is exactly reproduced. Let $\mathbf{r} \in \mathbb{R}^{K\times 1}$ denote the vector of r_j values. Then, by definition, we have

$$\mathbf{d}_0' := \sum_{j=1}^{K} c_j r_j \mathbf{g}_j = \mathbf{G}^{\mathrm{T}}\mathbf{r}.$$

To make the left hand side equal to $\mathbf{d}_0$, we have to solve this system of linear equations for $\mathbf{r}$ as $(\mathbf{G}^{\mathrm{T}})^{-1}\mathbf{d}_0$. In some rare cases $\mathbf{G}$ may be non-invertible. It can be shown that even in this case $\mathbf{d}_0$ is in the image of matrix $\mathbf{G}$ (the proof is omitted here due to the lack of space). Summarizing the above construction in a nutshell: rating all items with c_0 confidence and r_0 as rating value is equivalent to rating the synthetic items $\mathbf{g}_j$ ($j = 1, ..., \ldots, K$) with $c_j = 1$ confidence and r_j as rating value.

Using synthetic examples.

Now we describe how IALS1 uses the synthetic examples. To recompute $\mathbf{p}_u$, we assume that the user u rated the following items:

- for $j = 1, \ldots, K$, she rated $\mathbf{g}_j$ as r_j, with $c_j = 1$ confidence level (synthetic negative implicit feedback examples);
- for $i : (u, i) \in \mathcal{R}$, she rated $\mathbf{q}_i$ as r_0, with $-c_0$ confidence level (negative implicit feedback cancelation examples);
- for $i : (u, i) \in \mathcal{R}$, she rated $\mathbf{q}_i$ as r_{ui}, with c_{ui} confidence level (aggregation of positive implicit feedbacks).

We apply RR1 for 1 cycle with these examples, $\mathbf{p}_u$ is both the input and the output weight vector for RR1.

Handling regularization.

Regularization (usually) means that the diagonal of $\mathbf{A}$ is increased by a regularization factor λ, which can be also user-dependent: λ_u. In the context of IALS1, regularization can be handled easily: assume that the regularization factor for user u is λ_u, i.e. we try to minimize the following cost function:

$$\sum_{(u,i)\in\mathcal{R}^{\pm}} c_{ui}\left(r_{ui} - \mathbf{p}_u^{\mathrm{T}}\mathbf{q}_i\right)^2 + \sum_{u=1}^{N} \lambda_u \mathbf{p}_u^{\mathrm{T}}\mathbf{p}_u + \sum_{i=1}^{M} \lambda_i \mathbf{q}_i^{\mathrm{T}}\mathbf{q}_i.$$

When optimizing $\mathbf{p}_u$, we can create K synthetic regularization examples with rating value 0 and confidence level 1. Let the k-th such synthetic example be the k-th row of $\sqrt{\lambda_u}\mathbf{I}$. If the user rates all these K examples with confidence level 1 and rating value 0, the diagonal of $\mathbf{A}_u$ is increased exactly by $\lambda_u\mathbf{I}$.

Now we summarize the steps of recomputing $\mathbf{p}_u$, assuming user-dependent regularization:

- for $j = 1, \ldots, K$, she rated $\mathbf{g}_j$ as r_j, with $c_j = 1$ confidence level (synthetic negative implicit feedback examples);
- for $i : (u, i) \in \mathcal{R}$, she rated $\mathbf{q}_i$ as r_0, with $-c_0$ confidence level (negative implicit feedback cancelation examples);
- for $i : (u, i) \in \mathcal{R}$, she rated $\mathbf{q}_i$ as r_{ui}, with c_{ui} confidence level (aggregation of positive implicit feedbacks);
- for $j = 1, \ldots, K$, she rated the j-th row of $\sqrt{\lambda_u}\mathbf{I}$ as 0, with 1 confidence level (regularization examples).

We apply RR1 for 1 cycle with these examples on $\mathbf{p}_u$ being both the input and the output of the algorithm.

A few remarks on regularization:

- We can achieve the same results by not creating these synthetic regularization examples, but calling RR1 with λ_u as a regularization factor. In other words: we can exploit the sparseness of the regularization examples.
- If the value of λ_u does not depend on u, we can merge these synthetic regularization examples into the $\mathbf{A}_0$ and $\mathbf{d}_0$ variables, and skip the regularization step when optimizing $\mathbf{p}_u$.
- We can reduce the number of examples in RR1 cycles by merging the negative implicit feedback cancelation example and the aggregation of positive implicit feedbacks to a single positive difference example. This can be both used in the regularized and in the non-regularized version. Practically the 2nd and 3rd subsets of examples are unified as
  - for $i : (u, i) \in \mathcal{R}$, she rated $\mathbf{q}_i$ as $\frac{-c_0 r_0 + c_{ui} r_{ui}}{-c_0 + c_{ui}}$, with $-c_0 + c_{ui}$ confidence (positive difference examples)

Computational complexity.

In the beginning of the $\mathbf{P}$-step, we need to compute $\mathbf{G}$ and $\mathbf{r}$, which is $O(MK^2 + K^3)$. For each user, optimizing $\mathbf{p}_u$ requires $O(K(K + n_u + n_u)) = O(K^2 + Kn_u)$ steps. If we sum it for each user, we get $O(NK^2 + K|\mathcal{R}|)$. Thus, the $\mathbf{P}$-step requires $O(MK^2 + K^3 + NK^2 + K|\mathcal{R}|)$ time. If we assume that $M > K$, which means that we have fewer features than items, then this formula reduces to:

$$O(MK^2 + NK^2 + K|\mathcal{R}|),$$

which is very favorable compared to the $O(K^2|\mathcal{R}|)$ of IALS, since $M + N \ll |\mathcal{R}|$.

4. EXPERIMENTS

4.1 Datasets

We evaluated ALS1 on the Netflix Prize (NP) dataset [4] and IALS1 on a proprietary dataset of one of our customers.

4.1.1 *The Netflix Prize dataset*

The NP dataset contains 100 480 507 ratings (i.e. (u, i, r_{ui}) triplets) of 480 189 users on 17 770 movies, on a 1 to 5 integer scale. Ratings are provided as a standard train-test split. The test set contains 1 408 395 ratings and is called Probe and contains newer ratings for each user than the

Train. In our experiments we used our *Probe10* set[2] that contains only a 1/10 subset of the Probe ratings (140 840 ratings), and put the rest of Probe into Train. Probe10 approximates very well the withheld Quiz set [11], in the sense that predictors evaluated on both produce very similar results; the difference is almost always less than 2 bps;[3] due to this Probe10 has been also used by others in the NP community. Before experimentation, we subtracted the global mean, 3.6043, from all ratings.

4.1.2 The repetitive implicit feedback dataset

We applied a de-identified repetitive implicit feedback (RIF) dataset for implicit feedback experiments.[4] The database consists of 2 parts; the first one contains 9 617 414 implicit feedbacks from 215 630 users on 73 863 items created in a period of 182 days (the very same user can have several implicit feedbacks on the same item), the second one provides time information about availability of items. We partitioned the set of implicit feedbacks into a train set (181 days, 9 439 863 events) and a test set (1 day, 55 711 events) for the evaluation of IALS and IALS1.

4.2 Results

All experiments were performed on an Intel Core2 Quad Q9300 cpu on 2.5GHz, using only 1 core.

4.2.1 Netflix Prize dataset

We evaluated the ALS and the ALS1 algorithms on the Netflix Prize dataset, optimizing the following cost function:

$$\sum_{(u,i)\in\mathcal{R}} \left((\mathbf{p}_u^{\mathrm{T}}\mathbf{q}_i - r_{ui})^2 + \lambda^{(p)}\mathbf{p}_u^{\mathrm{T}}\mathbf{p}_u + \lambda^{(q)}\mathbf{q}_i^{\mathrm{T}}\mathbf{q}_i \right),$$

where $\lambda^{(p)} = 2$ and $\lambda^{(q)} = 0.001$, and we restricted the first column of $\mathbf{P}$ and the second column of $\mathbf{Q}$ to be 1.

We varied the number of features, K, and measured the RMSE on the test set after each epoch. Table 1 summarizes the results after 10 and 25 epochs. We also report the time needed to train the model. The RMSE of ALS and ALS1 are more similar after 25 than after 10 epochs: the largest difference is only 4 bps. However, ALS1 requires much less time for training: as K grows, the gap becomes larger. For the typical choice of $K = 50$, the training with ALS1 for 25 epochs requires less than an hour, while ALS needs more than 7 hours. When K is 100, ALS1 is already 17 times faster. We were further able to run ALS1 with $K = 1000$ faster than ALS with $K = 100$.

From these tables we can conclude, that on the Netflix Prize dataset, ALS1 is about an order of magnitude faster than ALS at the usual K values, while offering almost the same RMSE. On the other hand if that small difference in RMSE matters, we can run ALS1 with somewhat more factors to catch up with the accuracy of ALS.

Figure 1 shows the results measured in each epoch. We depicted only the best RMSE results achieved with both ALS and ALS1 under the same amount of time, independently from the number of epochs and the value of K. Both obtained curves are monotone decreasing. The curve of ALS1 is always under ALS which means that:

- ALS1 can reach the same RMSE as ALS much faster.
- ALS1 can reach significantly better RMSE than ALS under the same amount of training time.

Consequently, ALS1 offers a superior trade-off possibility between running time and performance compared to ALS.

	ALS		ALS1	
K	RMSE	time	RMSE	time
5	0.9391	389	0.9394	305
10	0.9268	826	0.9281	437
20	0.9204	2288	0.9222	672
50	0.9146	10773	0.9154	1388
100	0.9091	45513	0.9098	2653
200	0.9050	228 981	0.9058	6308
500	0.9027	2 007 451	0.9032	22070
1000	N/A	N/A	0.9025	44345

	ALS		ALS1	
K	RMSE	time	RMSE	time
5	0.9386	980	0.9390	764
10	0.9259	2101	0.9262	1094
20	0.9192	5741	0.9196	1682
50	0.9130	27500	0.9134	3455
100	0.9078	115 827	0.9079	6622
200	0.9040	583 445	0.9041	15836
500	0.9022*	4 050 675*	0.9021	55054
1000	N/A	N/A	0.9018	110904

Table 1: Probe10 RMSE of ALS and ALS1 on the Netflix dataset, after 10 (upper) and 25 (lower) epochs and the same K. Reported training times are in seconds
*** ran only for 20 epochs.**

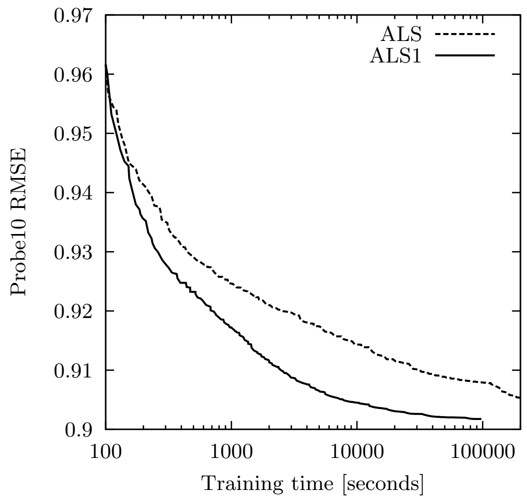

Figure 1: Comparison of ALS and ALS1

4.2.2 RIF dataset, ARP

We performed the evaluation of implicit feedback algo-

[2]A Perl script is available at our homepage, `gravityrd.com`, which selects the Probe10 from the original Netflix Probe set to ensure repeatability.
[3]1 bps = 0.0001 RMSE
[4]The terms of the agreement allows us to publish the de-identified dataset with the proprietary information removed, but the owner wanted to remain anonym.

rithms on the RIF dataset. We optimized the following cost function:

$$\sum_{(u,i)\in\mathcal{R}^{\pm}} c_{ui}\cdot(\mathbf{p}_u^{\mathrm{T}}\mathbf{q}_i - r_{ui})^2 + \lambda_u \sum_{u=1}^{N}\mathbf{p}_u^{\mathrm{T}}\mathbf{p}_u + \lambda_i \sum_{i=1}^{M}\mathbf{q}_i^{\mathrm{T}}\mathbf{q}_i,$$

where $c_{ui} = 1 + \alpha \cdot r_{ui}$ [7], $\alpha = 200$, $\lambda_u = \lambda_i = 1000$.

We the used average relative position (ARP) as the evaluation metric for this implicit feedback dataset, which is defined as follows.

For a given user u in the test set, the recommendation algorithm has to order all the recommendable items. This ordering can be based on the $\hat{r}_{ui} \in \mathbb{R}$ prediction values: items with highest $\hat{r}_{ui}$ are ranked first. For user u, the set of items are divided into two parts: relevant and non-relevant ones for the user. Assume that the recommendable items are indexed by i ranging from 1 to M. Let $r_{ui} \in \{0,1\}$ denote whether an item is relevant to the user or not, e.g. user will view that item during the test period or not. Then the position for item i relevant to user u is defined as:

$$\mathrm{pos}_{ui} = |\{j : \quad r_{uj} = 0 \wedge \hat{r}_{uj} > \hat{r}_{ui}\}|$$

The relative position is defined as:

$$\mathrm{rpos}_{ui} = \frac{\mathrm{pos}_{ui}}{|\{j : \quad r_{uj} = 0\}|}$$

For example, if we have 103 items, only 3 of them (indexed by 1,2 and 3) are relevant for user u, and the ranking algorithm puts them at the 1st, 20th and 54th position, then the relative positions are $\mathrm{rpos}_{u1} = 1/100$, $\mathrm{rpos}_{u2} = 20/100$ and $\mathrm{rpos}_{u3} = 54/100$. This is a normalized measure.

The average relative position is defined as:

$$\mathrm{ARP} = \frac{\sum_{(u,i)\in\mathcal{R}} \mathrm{rpos}_{ui}}{|\mathcal{R}|}, \qquad (14)$$

where the summation is taken over each positive implicit feedback of the test period. We remark that during training, each positive implicit feedback of the test set (i.e. $r_{ui} = 1$) must be considered as it would be negative implicit feedback (i.e. $r_{ui} = 0$) to avoid information leakage. We remark that a very similar measure was proposed in [7, 8], indeed ARP can be seen as the formal definition of the rank measure proposed in these works.

We experimented on the RIF dataset with various K values to compare the performance of IALS and IALS1 algorithms.

Table 2 contains comparative results of the evaluation after 10 and 20 epochs. With increasing K the difference between the training time of IALS and IALS1 gets larger. When K is 100, IALS1 is 10 times faster than IALS. When we compare a results of IALS to the results of IALS1 on the next K values (at 20 epochs9, we can see that IALS1 is both faster and more accurate in these comparisons, especially when K is large. For example, IALS1 with $K = 500$ can reach better performance in a shorter time than IALS with $K = 250$.

Figure 2 compares the time–accuracy trade-off of the two algorithms. We used the same consideration to depict the curves as for Figure 1. The curve of IALS1 is always located under IALS, so we can draw similar conclusions as before, i.e. IALS1 offers superior trade-off possibility between running time and performance compared to IALS.

The curves of IALS1 and IALS have similar characteristics: steep start, flat in the middle, and gets steep again at the end. When examining the curves from left to right, both curves reach a point when their gradient decreases and then the two curves become closer. Usually the ARP is better for larger K values. However, in the range of $K = 100$–300 (on the time scale between 2000 and 20000 seconds) this does not hold (the flat middle), and consequently the two curves get close to each other. This irregular behavior may be due to overlearning, perhaps the regularization scheme should be reconsidered.

	IALS		IALS1	
K	ARP	time	ARP	time
5	0.1903	76	0.1890	56
10	0.1584	127	0.1598	67
20	0.1429	322	0.1453	104
50	0.1342	1431	0.1366	262
100	0.1328	5720	0.1348	680
250	0.1316	46472	0.1329	3325
500	0.1282	244 088	0.1298	12348
1000	N/A	N/A	0.1259	52305

	IALS		IALS1	
K	ARP	time	ARP	time
5	0.1903	153	0.1898	112
10	0.1578	254	0.1588	134
20	0.1427	644	0.1432	209
50	0.1334	2862	0.1344	525
100	0.1314	11441	0.1325	1361
250	0.1313	92944	0.1311	6651
500	N/A	N/A	0.1282	24697
1000	N/A	N/A	0.1242	104 611

Table 2: ARP of IALS and IALS1 with different number of factors on the RIF dataset, after 10 (upper) and 20 (lower) epochs. Reported times are in seconds.

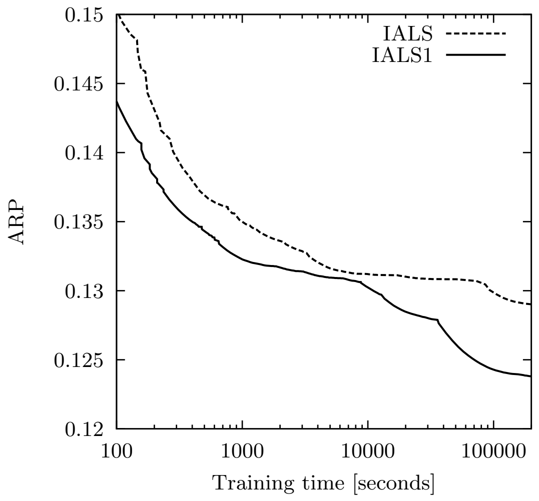

Figure 2: Comparison of IALS and IALS1

5. RELATED WORK

In the NP competition several authors tried to squeeze better results from the same algorithm using many parameters [2, 11, 13]. For example in [13], the authors report on the use of a parallel Matlab on a Linux cluster with 30 machines to run an ALS variant with $K = 1000$ and achieve Quiz RMSE = 0.8985. Note that we run ALS1 with $K = 1000$ on a single machine.

Training one feature at a time has also been used for MF algorithms. In the seminal work of Simon Funk [5], an MF variant is proposed using such a feature training; each feature is trained consecutively until overlearning is detected. There are three important differences between the methods: (1) Funk's algorithm does not optimize old features, just introduce new ones one-by-one; (2) his algorithm trains one feature per epoch, thus it requires many epochs (many re-iteration through the ratings); (3) his algorithm applies gradient descent.

In [1], another ALS variant is proposed, which shares many characteristics with Funk's one: (1) it does not optimize old features; (2) trains one feature per epoch; (3) there is no explicit cost function, the computation of the residual of old features is rather complicated.

6. CONCLUSIONS, FUTURE WORK

In this paper we presented novel and fast ALS variants for recommendation problems. The article has three main contributions:

- The proposed algorithms (ALS1 and IALS1) approximately minimize the same target functions as their counterparts, ALS and IALS, while they have better trade-off between running time and accuracy. These algorithms enable us to compute matrix factorization with much larger number of factors (e.g. 1000).
- ALS does not need to be used with traditional ridge regression (RR), we can apply approximate methods that optimize on the formerly computed values of $\mathbf{P}$ and $\mathbf{Q}$. Here we proposed to use RR1 instead of RR, but other approximations may work as well.
- IALS: we can replace the precomputed covariance matrix $\mathbf{A}_0$ with synthetic examples. Thus, the negative implicit feedback on a huge number of items can be replaced with a precomputed feedback on K synthetic examples. This reduction allows for the use of other methods, such as e.g. non-linear ones.

Since ALS is a general method for matrix factorization, as future work we intend to experiment with the proposed ALS variants on other application domains different from collaborative filtering. We are also willing to investigate the idea of IALS1 when the preference/confidence matrices of negative implicit feedback are of rank K_2. The proposed ALS1 and IALS1 store only the diagonal of the covariance matrix. We may relax this restriction and store data also in the box-diagonal. This leads to multivariate regression problems but with small number of variables. At IALS1 gradient descent method can replace RR1, offering the same time complexity. Here, in order to avoid negative learning rate caused by negative confidence values, one has to use merged single positive difference examples, as proposed in our last remark at IALS1.

Acknowledgement

D. Tikk was supported by the Alexander-von-Humboldt Foundation.

7. REFERENCES

[1] R. Bell, Y. Koren, and C. Volinsky. Modeling relationships at multiple scales to improve accuracy of large recommender systems. In *KDD-07, 13th ACM Int. Conf. on Knowledge Discovery and Data Mining*, pages 95–104, New York, NY, USA, 2007. ACM.

[2] R. M. Bell and Y. Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *ICDM-07, 7th IEEE Int. Conf. on Data Mining*, pages 43–52, Omaha, Nebraska, USA, 2007.

[3] R. M. Bell, Y. Koren, and C. Volinsky. The BellKor solution to the Netflix Prize. Technical Report, AT&T Labs Research, 2007.

[4] J. Bennett and S. Lanning. The Netflix Prize. In *KDD Cup Workshop at KDD-07, 13th ACM Int. Conf. on Knowledge Discovery and Data Mining*, pages 3–6, San Jose, California, USA, 2007.

[5] S. Funk. Netflix update: Try this at home, 2006. `http://sifter.org/~simon/journal/20061211.html`.

[6] J. Herlocker, J. A. Konstan, and J. Riedl. An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. *Inf. Retr.*, 5(4):287–310, 2002.

[7] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM-08, 8th IEEE Int. Conf. on Data Mining*, pages 263–272, Pisa, Italy, 2008.

[8] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD-08: 13th ACM Int. Conf. on Knowledge Discovery and Data Mining*, pages 426–434, New York, NY, USA, 2008.

[9] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *KDD Cup Workshop at KDD-07, 13th ACM Int. Conf. on Knowledge Discovery and Data Mining*, pages 39–42, San Jose, California, USA, 2007.

[10] I. Pilászy and D. Tikk. Computational complexity reduction for factorization-based collaborative filtering algorithms. In *EC-Web-09: 10th Int. Conf. on Electronic Commerce and Web Technologies*, pages 229–239, 2009.

[11] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Scalable collaborative filtering approaches for large recommender systems. *Journal of Machine Learning Research*, 10:623–656, 2009.

[12] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Investigation of various matrix factorization methods for large recommender systems. In *2nd Netflix-KDD Workshop at KDD-08: 14th ACM Int. Conf. on Knowledge Discovery and Data Mining*, Las Vegas, NV, USA, August 24, 2008.

[13] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan. Large-scale parallel collaborative filtering for the Netflix Prize. In *AAIM-08: 4th Int. Conf. on Algorithmic Aspects in Information and Management*, pages 337–348, Berlin, Heidelberg, 2008. Springer-Verlag.