2008 IEEE International Conference on Data Mining Workshops

Investigation of Various Matrix Factorization Methods for Large Recommender Systems

Gábor Takács*
Széchenyi István University
Egyetem tér 1.
Győr, Hungary
gtakacs@sze.hu

István Pilászy, Bottyán Németh, Domonkos Tikk
Budapest University of Technology and Economics
Magyar Tudósok krt. 2.
Budapest, Hungary
{pila,bottyan,tikk}@tmit.bme.hu

Abstract

Matrix Factorization (MF) based approaches have proven to be efficient for rating-based recommendation systems. In this work, we propose several matrix factorization approaches with improved prediction accuracy. We introduce a novel and fast (semi)-positive MF approach that approximates the features by using positive values for either users or items. We describe a momentum-based MF approach. A transductive version of MF is also introduced, which uses information from test instances (namely the ratings users have given for certain items) to improve prediction accuracy. We describe an incremental variant of MF that efficiently handles new users/ratings, which is crucial in a real-life recommender system. A hybrid MF–neighbor-based method is also discussed that further improves the performance of MF. The proposed methods are evaluated on the Netflix Prize dataset, and we show that they can achieve very favorable Quiz RMSE (best single method: 0.8904, combination: 0.8841) and running time.

1 Introduction

Recommender systems attempt to profile user preferences over items and models the relation between users and items. The task of recommender systems is to recommend items that fit the user's taste, in order to help the user in selecting/purchasing items from an overwhelming set of choices. Such systems have great importance in applications such as e-commerce, subscription-based services, information filtering, etc. Recommender systems providing personalized suggestions greatly increase the likelihood of a customer making a purchase compared to unpersonalized ones. Personalized recommendations are especially important in markets where the variety of choices is large, the taste of the customer is important, and typically the price of the items is modest. Typical areas of such services are mostly related to art (esp. books, movies, music), fashion, food & restaurants, gaming & humor.

With the growing significance of e-commerce, an increasing number of web-based merchant and rental services use recommender systems. Some of the major participants of e-commerce web, like Amazon.com and Netflix, successfully apply recommender systems to deliver automatically generated personalized recommendation to their customers. The importance of a good recommender system was recognized by Netflix, which led to the announcement of the Netflix Prize (NP) competition to motivate researchers to improve the accuracy of their recommender system called Cinematch.

The approach which makes use of only user activities of the past[1] (e.g. transaction history or user satisfaction expressed in rating) is termed *collaborative filtering* (CF). The NP contest focuses on the case when users express their opinion of items by means of ratings. In this framework, the user first provides ratings of some items usually on a discrete numerical scale, and the system then recommends other items based on ratings similar users have already provided.

Matrix factorization based techniques have proven to be efficient in recommender systems (see Section 1.1) when predicting user preferences from known user-item ratings. The main contribution of this work is a number of novel MF based algorithms that are accurate in predicting user ratings, and provide scalable solutions for large-scale recommender systems. In particular, we present

- an MF with biases, which is currently our best performing approach.
- a novel and fast (semi-)positive MF approach that ap-

*All authors are also affiliated with Gravity Research & Development Ltd., H-1092 Budapest, Kinizsi u. 11., Hungary, info@gravitrd.com

[1]In contrast to the *content-based approaches* which use also demographic data to profile users.

proximates the factors by using positive values for either users or items;
- a momentum-based MF approach;
- a transductive version of MF that makes use of information from test instances (namely the ratings users have given for certain items) to improve prediction accuracy;
- an incremental variant of MF that efficiently handles new users/ratings (this is crucial in a real-life recommender systems);
- a hybrid MF–neighbor based method is introduced, which improves the accuracy of MF considerably.

The proposed methods were evaluated on the Netflix Prize problem, but this does not limit their applicability to this specific dataset.

1.1 Related Work

The first works on the field of CF have been published in the early 1990s. The Tapestry system [7] used collaborative filtering to filter mails simultaneously from several mailing lists based on the opinion of the community on readings. Over the last broad decade many CF algorithms have been proposed that approach the problem by different techniques, including similarity/neighborhood based approaches [11, 14], Bayesian networks [6], restricted Boltzman machines (RBM) [13], and various matrix factorization techniques [8, 15].

The NP competition boosted the interest in CF, and yielded a number of related publications. We should here mention the NP related 1st Netflix-KDD Workshop in 2007 [4], which brought together top contenders of the contest. The members of BellKor/KorBell team[2] presented an improved neighborhood based approach in [2], which removes the global effect from the data—can be considered as normalization—to improve the accuracy of similarity based interpolative predictions. Paterek applied successfully various matrix factorization techniques [10] by adding biases to the regularized MF, postprocessing the residual of MF with kernel ridge regression, using a separate linear model for each movie, and by decreasing the parameters in regularized MFs. Kurucz et al showed the application of expectation maximization based MF methods for NP [9].

Our methods are different from the above ones in various aspects. BellKor uses alternate least squares, but we use incremental gradient descent method at weight updates. Their method does not use the chronological order of ratings, while we exploit this information in our approaches. The accuracy of their published MF variants is inferior to ours. They use only positive and normal MFs, while we propose the semi-positive version of MF algorithm. The learning of BellKor's positive MF is more complicated and consequently significantly slower than ours, but this complexity does not yield improvement on the accuracy. Paterek apply a different learning scheme that compares unfavorable to ours in terms of speed and accuracy. We point out that this difference result in faster training and better accuracy of our MF methods. Other differences are that Paterek uses less meta-parameters for his MF methods. The idea of using test data has also appeared at various authors for different approaches: RBM in [13], LM, NSVD1 and NSVD2 in [10]. Our presented approaches differ slightly from known ones but these modifications are together important: simultaneous feature training, regularization, bias features, early stopping criteria, incremental gradient descent training algorithm, and date based ordering of the ratings of users.

[2]Winner of the Progress Prize 2007, awarded to the leading team at the one-year anniversary of NP.

2 Problem definition

We define the problem of collaborative filtering in the following setting. A set of I users and a set of J items are given. A rating record is a quadruple (i, j, d_{ij}, x_{ij}) representing that user i rated item j on date d_{ij} as x_{ij}, where $i \in \{1, \ldots, I\}$, $j \in \{1, \ldots, J\}$, $d_{ij} \in \mathcal{D}$ the ordered set of possible dates, and $x_{ij} \in \mathcal{X} \subset \mathbb{R}$. Typical rating values can be binary ($\mathcal{X} = \{0, 1\}$), integers from a given range (e.g. $\mathcal{X} = \{1, 2, 3, 4, 5\}$), or real numbers of a closed interval (e.g. $\mathcal{X} = [-1, 10]$). We assume that a given user can rate a given item at most once. This justifies the use of subscripts ij for dates and rating values. We are given a finite set of rating records, $\mathcal{T}$, which are used for the training. We refer to the set of all known (i, j) pairs in $\mathcal{T}$ as $\mathcal{R}$. Note that typically $|\mathcal{R}| \ll |I| \cdot |J|$, because each user rates only a few items.

The rating values can be organized in a rating matrix $\mathbf{X}$ where elements indexed by $(i, j) \notin \mathcal{R}$ are unknown. In this paper we adopt the evaluation measure of NP contest, the root mean squared error (RMSE), which is defined as:

$$RMSE(T) = \sqrt{\frac{1}{|T|} \sum_{(i,j) \in T} (\hat{x}_{ij} - x_{ij})^2}, \qquad (1)$$

where T (i, j) contains user-item pairs on which the ratings are predicted. The accuracy of predictors are evaluated on a validation set $\mathcal{V}$; naturally the ratings of $\mathcal{V}$ is not used at the creation of the predictor. Since in recommender systems the goal is to predict the user preferences from past ratings, the ratings of $\mathcal{T}$ precedes the ratings of $\mathcal{V}$ in time.

From now on, without loss of generality, we use terms item and movie as synonyms. At the prediction of a given rating we refer to the user as *active user*, and to the movie as *active movie*. Superscript „hat" denotes the prediction of the given quantity, that is, $\hat{x}$ is the prediction of x.

3 Matrix factorization methods

MF techniques approximate $\mathbf{X}$ as a product of two much smaller matrices:

$$\mathbf{X} \approx \mathbf{U}\mathbf{M}, \tag{2}$$

where $\mathbf{U}$ is an $I \times K$ and $\mathbf{M}$ is a $K \times J$ matrix.

3.1 Basic MF

In the case of the given problem, $\mathbf{X}$ has many unknown elements which cannot be treated as zero. For this case, the approximation task can be defined as follows. Let $\mathbf{U} \in \mathbb{R}^{I\times K}$ and $\mathbf{M} \in \mathbb{R}^{K\times J}$. Let u_{ik} denote the elements of $\mathbf{U}$, and m_{kj} the elements of $\mathbf{M}$. Let $\mathbf{u}_i^T$ denote a row of $\mathbf{U}$, and $\mathbf{m}_j$ a column of $\mathbf{M}$. Then:

$$\hat{x}_{ij} = \sum_{k=1}^{K} u_{ik} m_{kj} = \mathbf{u}_i^T \mathbf{m}_j \tag{3}$$

$$e_{ij} = x_{ij} - \hat{x}_{ij} \quad \text{for } (i,j) \in \mathcal{R}$$

$$e'_{ij} = \frac{1}{2} e_{ij}^2 \tag{4}$$

$$\text{SSE} = \sum_{(i,j)\in\mathcal{R}} e_{ij}^2, \qquad \text{SSE}' = \frac{1}{2}\text{SSE} = \sum_{(i,j)\in\mathcal{R}} e'_{ij}$$

$$\text{RMSE} = \sqrt{\text{SSE}/|\mathcal{R}|} \tag{5}$$

$$(\mathbf{U}^*, \mathbf{M}^*) = \underset{(\mathbf{U},\mathbf{M})}{\arg\min}\, \text{SSE}' = \underset{(\mathbf{U},\mathbf{M})}{\arg\min}\, \text{SSE} = \underset{(\mathbf{U},\mathbf{M})}{\arg\min}\, \text{RMSE} \tag{6}$$

Here $\hat{x}_{ij}$ denotes how the i-th user would rate the j-th movie, according to the model, e_{ij} denotes the training error on the (i,j)-th rating, and SSE denotes the sum of squared training errors. Eq. (6) states that the optimal $\mathbf{U}$ and $\mathbf{M}$ minimizes the sum of squared errors only on the known elements of $\mathbf{X}$.

In order to minimize RMSE, which is equivalent to minimize SSE$'$, we have applied a simple incremental gradient descent method to find a local minimum of SSE$'$, where one gradient step intend to decrease the square of prediction error of only one rating, or equivalently, either e'_{ij} or e_{ij}^2. Suppose we are at the (i,j)-th training example, x_{ij} and its approximation $\hat{x}_{ij}$ is given.

We compute the gradient of e'_{ij}:

$$\frac{\partial}{\partial u_{ik}} e'_{ij} = -e_{ij} \cdot m_{kj}, \quad \frac{\partial}{\partial m_{kj}} e'_{ij} = -e_{ij} \cdot u_{ik}. \tag{7}$$

We update the weights in the direction opposite of the gradient:

$$u'_{ik} = u_{ik} + \eta \cdot e_{ij} \cdot m_{kj}, \quad m'_{kj} = m_{kj} + \eta \cdot e_{ij} \cdot u_{ik}. \tag{8}$$

that is, we change the weights in $\mathbf{U}$ and $\mathbf{M}$ to decrease the square of actual error, thus better approximating x_{ij}. Here η is the learning rate. We refer to this method as Basic MF.

3.2 Regularized MF

Consider the following case: let $K = 2$, $m_{71} = 1$, $m_{72} = 0$, $m_{81} = 1$, $m_{82} = 0.1$, $x_{67} = 4$, $x_{68} = 3$, $|\{j : (i,j) \in \mathcal{R}, i = 6\}| = 2$. In other words, we have two features, and the 6th user rated only two very similar movies, the 7th and the 8th, as a 4 and a 3. In this case, according to eq. (6), the optimal user features are $u_{61} = 4$, $u_{62} = -10$, which perfectly describe the ratings of this user: $\hat{x}_{67} = 4, \hat{x}_{68} = 3$.

Apparently such large feature values ($x_{62} = -10$) should be avoided. The common way of overcoming this consists in applying regularization by penalizing the square of the Euclidean norm of weights, which results in a new optimization problem:

$$e'_{ij} = (e_{ij}^2 + \lambda \cdot \mathbf{u}_i^T \cdot \mathbf{u}_i + \lambda \cdot \mathbf{m}_j^T \cdot \mathbf{m}_j)/2 \tag{9}$$

$$\text{SSE}' = \sum_{(i,j)\in\mathcal{R}} e'_{ij} \tag{10}$$

$$(\mathbf{U}^*, \mathbf{M}^*) = \underset{(\mathbf{U},\mathbf{M})}{\arg\min}\, \text{SSE}' \tag{11}$$

Note that minimizing SSE$'$ is no longer equivalent to minimizing SSE. Similar to the Basic MF approach, we compute the gradient of e'_{ij}, and update the weights in the direction opposite of the gradient:

$$\frac{\partial}{\partial u_{ik}} e'_{ij} = -e_{ij} \cdot m_{kj} + \lambda \cdot u_{ik}, \quad \frac{\partial}{\partial m_{kj}} e'_{ij} = -e_{ij} \cdot u_{ik} + \lambda \cdot m_{kj}. \tag{12}$$

$$u'_{ik} = u_{ik} + \eta \cdot (e_{ij} \cdot m_{kj} - \lambda \cdot u_{ik}), \tag{13}$$

$$m'_{kj} = m_{kj} + \eta \cdot (e_{ij} \cdot u_{ik} - \lambda \cdot m_{kj}) \tag{14}$$

For the learning algorithm used in Regularized MF, see Algorithm 1.

We remark that after the learning phase, each value of $\mathbf{X}$ can be computed easily using eq. (3), even the “unseen” values. In other words, the model ($\mathbf{U}^*$, $\mathbf{M}^*$) provides a description of how an arbitrary user would rate any movie.

3.3 BRISMF

We found a simple way to boost the performance of Regularized MF, by fixing the first column of $\mathbf{U}$ and the second row of $\mathbf{M}$ to the constant value of 1. Under the expression “fixing to a constant value” we mean not to apply eqs. (13)–(14) when updating u_{i1} and m_{2j}, and in the initialization, to assign them the constant value instead of random values. The pair for these features (m_{1j} and u_{i2}) can serve as a bias feature. This simple extension speeds up the training phase and yields a more accurate model with better generalization performance. We refer to this method as BRISMF

```
    Input: $\mathcal{R}, \mathbf{X}, \eta, \lambda, \mathrm{T}, \mathcal{V}$
    Output: $\mathbf{U}^*, \mathbf{M}^*$
1   Initialize $\mathbf{U}$ and $\mathbf{M}$ with small random numbers.
2   loop until the terminal condition is met. One epoch:
3     iterate over each $(i, j)$ element of T. For $x_{ij}$:
4       compute $e'_{ij}$;
5       compute the gradient of $e'_{ij}$, according to (12);
6       for each $k \in \{1, \dots, K\}$
7         update the $i$-th row of $\mathbf{U}$ and the $j$-th column
8           of $\mathbf{M}$ according to (13)–(14);
9     calculate the RMSE on $\mathcal{V}$;
10    if the RMSE on $\mathcal{V}$ was better than in any prev.
11        epoch:
12      Let $\mathbf{U}^* = \mathbf{U}$ and $\mathbf{M}^* = \mathbf{M}$.
13    terminal condition: RMSE on $\mathcal{V}$ does not
    decrease
14      during two epochs.
15  end
```
Algorithm 1: Training algorithm for Regularized MF

that stands for Biased Regularized Incremental Simultaneous MF. In [16] the insertion of constant values into $\mathbf{U}$ and $\mathbf{M}$ was proposed. BRISMF is a special way of inserting constant values: all the inserted values are 1-s. We remark that the bias feature idea was mentioned also by Paterek in [10],

3.4 Semipositive and positive MF

The presented Regularized MF algorithm can assign not only positive but also negative feature values. Positive matrix factorization techniques have been succesfully applied in the field of CF [8]. We present a simple extension of Regularized MF that can give positive and semipositive factorizations. We talk about semipositive MF when exactly one of $\mathbf{U}$ and $\mathbf{M}$ contains both positive and negative values, and positive MF, when both contain only non-negative values.

The idea can be summarized as follows: for the (i, j)-th training example in a given epoch, if u_{ik} or m_{kj} becomes negative due to the application of eqs. (13)–(14), we reset it to 0. We describe the modified equations for the case when both user and movie features are required to be non-negative:

$$u'_{ik} = \max\{0, u_{ik} + \eta \cdot e_{ij} \cdot m_{kj} - \lambda \cdot u_{ik}\} \qquad (15)$$

$$m'_{kj} = \max\{0, m_{kj} + \eta \cdot e_{ij} \cdot u_{ik} - \lambda \cdot m_{kj}\} \qquad (16)$$

If we allow user features to be negative, we can simply use eq. (13) instead of eq. (15). Allowing only negative movie features can be treated similarly.

The presented approach can easily be extended to handle arbitrary box-constraints (i.e. prescribing a minimum and a maximum possible value for features e.g. 0 and 1). The handling of arbitrary box constraints can be thought of as a generalization of constant features, for example in the case of bias features, $1 \le u_{i1} \le 1$ and $1 \le m_{2j} \le 1$.

Though the (semi-)positive MF has no advantage over Regularized MF in the NP Problem, in real recommender systems the model can provide a better explanation of users' behaviour. When box constraints are 0 and 1, the feature values can be thought of as fuzzy membership values.

3.5 Applying momentum method

We can apply momentum method by a small modification of the original learning rule. In each learning step the modification of the weights are calculated not only from the actual gradient but from the last weight changes too. With the modification of (13) and (14) we get the following equations:

$$u_{ik}(t+1) = u_{ik}(t) + \Delta u_{ik}(t), \qquad (17)$$
$$\Delta u_{ik}(t) = \eta \cdot (e_{ij} \cdot m_{kj} - \lambda \cdot u_{ik}(t)) + \sigma \cdot \Delta u_{ik}(t-1),$$
$$m_{kj}(t+1) = m_{kj}(t) + \Delta m_{kj}(t), \qquad (18)$$
$$\Delta m_{kj}(t) = \eta \cdot (e_{ij} \cdot u_{ik} - \lambda \cdot m_{kj}(t)) + \sigma \cdot \Delta m_{kj}(t-1)$$

Here σ is the momentum factor. Though momentum MF is not among our best MFs, it blends well with other MF methods.

3.6 Retraining user features

The Regularized MF algorithm has a serious disadvantage: movie features change while we iterate through users. If the change is large, user features updated in the beginning of an epoch may be inappropriate at the end of the epoch.

To overcome this problem, we can completely recompute the user features after the learning procedure. This method can serve as an efficient incremental training method for recommender systems. The algorithm can be summarized as follows:

1. First training: apply an MF algorithm.
2. Let $\mathbf{M} = \mathbf{M}^*$. Initialize $\mathbf{U}$ randomly.
3. Second training: Apply the same learning procedure as in the first training step, with the following restrictions: skip the weight initialization step; do not change weights in $\mathbf{M}$, i.e. do not apply eq. (14) or its variants; store the optimal number of epochs, denote it n^*.

Note that this method can efficiently handle the addition of new users, or new ratings for an existing user, which is very important in a real recommender system: we do not apply the whole training procedure, just reset the weights of a certain user, and apply the second training procedure for n^* epochs, only for that certain user. Note also, that the second training procedure needs to iterate through the entire database, which requires slow disk access operation, only once (not n^* times), as the ratings of a user can be kept in memory and can be immediately re-used in the next epoch.

We found a simple modification of this algorithm very effective: in the second training, learn not only $\mathbf{U}$, but $\mathbf{M}$ as well. Though this is not useful for incremental learning in recommender systems anymore, it boosts the performance of MF.

3.7 Transductive MF

Transductive learning involves the use of test examples but not their labels. In the case of CF, this means that the algorithm "knows" what test examples the model will be applied for, i.e. the $(i, j) \in \mathcal{V}$ pairs, but not the corresponding x_{ij} values. For Restricted Boltzmann Machines there is a straightforward way to get a better model by incorporating the test set [13].

We have developed a Transductive MF that can incorporate $(i, j) \in \mathcal{V}$ information after the learning process. The idea behind the method is the following: suppose that user i has the following n ratings in the test set (test ratings) to be predicted: $x_{i1}, \ldots, x_{in}$. The user has a feature vector $(\mathbf{u_i})$. When we are at the j-th example, we can predict another feature vector based only on what other movies $(1 \le j' \le n, j' \ne j)$ are to be predicted. A proper linear combination – not depending on i or j – of the original user feature vector and this new feature vector can yield another feature vector for the prediction of x_{ij} that is better than the original one. Formally:

$$\mathbf{u}'_i(j) = \frac{1}{\sqrt{|j' : (i, j') \in \mathcal{R}| + 1}} \sum_{\substack{j''=1 \\ j'' \ne j}}^{n} \mathbf{m}_{j''}. \qquad (19)$$

$$\hat{x}'_{ij} = \hat{x}_{ij} + \nu \cdot \mathbf{u}'_i(j)^T \cdot \mathbf{m}_j = \mathbf{u}_i^T \cdot \mathbf{m}_j + \nu \cdot \mathbf{u}'_i(j)^T \cdot \mathbf{m}_j.$$

The attenuation factor in eq. (19) ensures that the more ratings a user has in a training set, the less information the test set provides, thus $\hat{x}'_{ij}$ will differ less from $\hat{x}_{ij}$. In practice, ν need not be determined: we can use $\hat{x}'_{ij}$ and $\mathbf{u}'_i(j)^T \cdot \mathbf{m}_j$ as two predictions for x_{ij}, and apply linear regression to get the best RMSE.

Though transductive methods cannot be applied directly in real applications, the transductive MF can boost up the performance of a predictor for the Netflix Prize problem.

3.8 Neighbor based correction of matrix factorization

The matrix factorization (MF) and the neighbor based (NB) approaches complement each other well:

- The MF approach views the data from a high level perspective. MF can identify the major structural patterns in the ratings matrix. An appealing property of MF is that it is able to detect the similarity between two items, even if no user rated both of them.
- The NB approach is more localized. It is typically good at modeling pairs of users/items and not so good at modeling interdependency within larger sets of users/items. NB methods are memory based, therefore they do not require any training.[3]

It is known that the combination of MF and NB can lead to very accurate predictions [2, 3]. However, conventional NB methods scale up poorly for large problems. The price of additional accuracy is the loss of scalability.

Here we propose a scalable scheme for using the MF and NB approaches together. The idea is that we improve an existing MF model (**U**,**M**) by adding an item neighbor based correction term to its answer in the prediction phase. The corrected answer for query (i, j) is the following:

$$\hat{r}_{ij} = \mathbf{u}_i^T \mathbf{m}_j + \gamma \frac{\sum_{k \in \mathcal{T}_i \setminus \{j\}} s_{jk} \left(\mathbf{u}_i^T \mathbf{m}_k - r_{ik} \right)}{\sum_{k \in \mathcal{T}_i \setminus \{j\}} s_{jk}},$$

where s_{jk} is the similarity between items j and k, and $\mathcal{T}_i$ is the set items rated by user i. The weight of the correction term γ can be optimized via cross-validation.

The similarity s_{jk} can be defined in many different ways. Here are two variants that proved to be useful for the Netflix Prize problem [5].

- (S1): Normalized scalar product based similarity.

$$s_{jk} = \left(\frac{\sum_{\ell=1}^{K} m_{\ell j} m_{\ell k}}{\sqrt{\sum_{\ell=1}^{K} m_{\ell j}^2} \cdot \sqrt{\sum_{\ell=1}^{K} m_{\ell k}^2}} \right)^{\alpha},$$

where α is the amplification parameter.

- (S2): Normalized Euclidean distance based similarity.

$$s_{jk} = \left(\frac{\sum_{\ell=1}^{K} (m_{\ell j} - m_{\ell k})^2}{\sqrt{\sum_{\ell=1}^{K} m_{\ell j}^2} \cdot \sqrt{\sum_{\ell=1}^{K} m_{\ell k}^2}} \right)^{\alpha}.$$

In both cases, the value s_{jk} can be calculated in O(K) time, thus $\hat{r}_{ij}$ can be calculated in O($K \cdot |\mathcal{T}_i|$). We remark that one can restrict to use only the top S neighbors of the queried item [2], however, it does not affect the time requirement, if we use the same function for s_{ij} and neighbor-selection.

Now let us comment in more details on the time and memory requirement of our NB corrected MF in comparison with the improved neighborhood based approach of

[3]However, it can be useful to precompute the similarity values to speed up the prediction phase.

BellKor [2], which can also be applied to further improve the results of an MF. For a give query, the time requirement of our method is O($K \cdot S$), while their method requires to solve a separate linear least squares problem with S variables, thus it is O(S^3). Memory requirements: for the i-th user, our method requires to store in the memory $\mathbf{u}_i$ and $\mathbf{M}$, that is O(KJ), while their approach necessitates to store the item-by-item matrix and the ratings of the user, which is O($J^2 + |\mathcal{T}_i|$). For all users, our method requires O($IK + KJ$) while their approach requires O($J^2 + |\mathcal{R}|$) memory.

Despite its simplicity, the effectiveness of our method is comparable with that of Bell and Koren's method, see Section 4.

The presented model can be seen as a nice unification of the MF and NB approaches. It is simple, scalable, and accurate. The training is identical to the regular MF training. The prediction consists of an MF and a NB term. The similarities used in the NB term need not to be precomputed and stored, because they can be calculated very efficiently from the MF model.

4 Experimental study

The experimentations have been performed on the Netflix Prize dataset, being currently the largest available one for the public. It contains about 100 million ratings from over 480k users on nearly 18k movies. For more information on the dataset see e.g. [3, 4]. We decided to validate our algorithm against the Netflix dataset since currently this is the most challenging problem for the CF community due to its particular properties (see e.g. [1] for a comprehensive summary of these properties), and the task of the Netflix Grand Prize.

In this experimentation section we evaluate the presented methods on a randomly selected 10% subset of the Probe set[4], which we refer to as Probe10. [5] We mention that we measured only a slight difference between the RMSE values on the two test sets: our Probe10 and Quiz[6]. For some selected methods, we also report on Quiz RMSE values. We performed all tests on a 2 GHz Intel Pentium M (Dothan) laptop with 1 GB RAM.

4.1 Parameter settings

All of the presented methods have many pre-defined parameters that greatly influences the result. A model and the corresponding parameter setting can be considered useful if either it results in a very accurate model (low test RMSE) or the model "blends well" i.e. improves the accuracy of the blended solution. For combining different models we applied ridge regression.[7]

We applied a very simple but effective parameter optimization algorithm. We initialize parameters randomly, then we select a single parameter to optimize. We assign n (typically 2) distinct random values to the parameter, and choose the best performing one by evaluating MF with this parameter setting. This method is performed systematically for all parameters one-by-one.

4.2 Results of Matrix Factorization

We ordered the training examples user-wise and then by date. By this we model that user's taste change in time, and the most recent taste is the dominant. This coincides with the creation of train-test split of NP dataset. We performed test runs with numerous parameters settings. We report on the actual settings of the parameters at each result. Naming convention for parameters: learning rate and regularization factor for users and movies ($\eta_u, \eta_m, \lambda_u, \lambda_m$); the corresponding variables of bias features ($\eta_{ub}, \eta_{mb}, \lambda_{ub}, \lambda_{mb}$); minimum and maximum weights in the uniform random initialization of $\mathbf{U}$ and $\mathbf{M}$: $w_{\underline{u}}, w_{\overline{u}}, w_{\underline{m}}, w_{\overline{m}}$; offset G to subtract from $\mathbf{X}$ before learning.

4.2.1 Comparing Regularized MF and BRISMF

We compare two MFs:

- a Regularized MF, which we name briefly as RegMF#0, with the following parameter settings: $K = 40, \eta = 0.01, \lambda = 0.01, w_{\underline{u}} = -w_{\overline{u}} = w_{\underline{m}} = -w_{\overline{m}} = -0.01$
- a BRISMF, which we name briefly as BRISMF#0, with the following parameter settings: $K = 40, \eta = 0.01, \lambda = 0.01, w_{\underline{u}} = -w_{\overline{u}} = w_{\underline{m}} = -w_{\overline{m}} = -0.01$

RegMF#0 reaches its optimal Probe10 RMSE in the 13th epoch: Probe10 RMSE is 0.9214, while these numbers for BRISMF#0 are: 10th and 0.9113, which is a 0.0101 improvement.

4.2.2 Changing the parameters of the BRISMF

Table 1 present the influence of η and λ on the Probe10 RMSE. Other parameters are the same as in BRISMF#0. Best result: $\eta = 0.007, \lambda = 0.005$. We refer to this MF in the following as BRISMF#1. The running time for this MF is only 14 minutes! The number of epochs to get this RMSE was 10. Note that running time depends only on K and the number of epochs to get the optimal Probe10 RMSE.

[4]Probe set: dedicated for testing purpose by Netflix with known ratings

[5]A Perl script will be made available at our home page upon acceptance, which selects the Probe10 from the original Netflix Probe set to ensure repeatability.

[6]Quiz set: to be predicted data, but only Netflix knows the ratings.

[7]The source code of our combination algorithm is publicly available at our web site.

Table 1. Probe10 RMSE / optimal number of epochs of the BRISMF for various η and λ values ($K = 40$)

η \ λ	0.005	0.007	0.010	0.015	0.020
0.005	0.9061	0.9079	0.9117	0.9168	0.9168
0.007	**0.9056**	0.9074	0.9112	0.9168	0.9169
0.010	0.9064	0.9077	0.9113	0.9174	0.9186
0.015	0.9099	0.9111	0.9152	0.9257	0.9390
0.020	0.9166	0.9175	0.9217	0.9314	0.9431

4.2.3 Subsampling users

On Figure 1 we demonstrate how the number of users (thus, the number of ratings) influences Probe10 RMSE and the optimal number of training epochs in case of BRISMF#1. Probe10 RMSE varies between 0.9056 and 0.9677, and the number of epochs between 10 and 26. The smaller the subset of users used for training and testing, the larger the Probe10 RMSE and the number of epochs. This means that time-complexity of MF is sublinear in the number of users, which is proportional to the number of ratings, the ratio is 209 for the Netflix dataset.

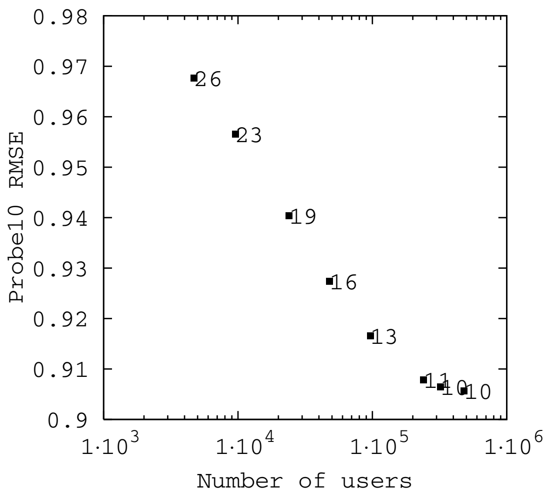

Figure 1. Effect of the number of users on Probe10 RMSE and on the optimal number of training epochs.

4.2.4 Retraining user features

We investigated with three parameter settings the effect of retraining user features: BRISMF#1, and the following two MFs:

- BRISMF#250: $K = 250$, $w_{\underline{u}} = -0.01$, $w_{\overline{u}} = -0.006$, $w_{\underline{m}} = -0.010$, $w_{\overline{m}} = 0.020$, $\eta_u = 0.008$, $\eta_{ub} = 0.016$, $\eta_m = 0.015$, $\eta_{mb} = 0.007$, $\lambda_u = 0.048$, $\lambda_m = 0.008$, $\lambda_{ub} = 0.019$, $\lambda_{mb} = 0, G = 0$.
- BRISMF#1000: the same as BRISMF#250, but $K = 1000$.

The results are summarized on Table 2. Both the simpler case (after the first learning step, reset **U** and retrain only **U**), and the advanced case (after the first learning step, reset **U** and retrain both **U** and **M**) are reported. We append letter "U" to the method name in the simpler case (i.e. BRISMF#1 becomes BRISMF#1U, etc.), and letters "UM" in the advanced case (BRISMF#1UM, etc.). We report the number of epochs required both in the first and the second training procedure (if available). Note, that in case of BRISMF#250 and BRISMF#1000, the retraining of user features greatly improved the performance of those MFs.

Table 2. Examining the effect of retraining user features. We report on Probe10 RMSE and some Quiz RMSE values.

Model	Epochs	Probe10	Quiz
BRISMF#1	10	0.9056	
BRISMF#1U	10+8	0.9072	
BRISMF#1UM	10+6	0.9053	
BRISMF#250	14	0.8961	0.8962
BRISMF#250U	14+8	0.8953	0.8954
BRISMF#250UM	14+7	0.8937	
BRISMF#1000	14	0.8938	0.8939
BRISMF#1000U	14+8	0.8936	
BRISMF#1000UM	14+8	0.8921	0.8918

4.2.5 Accurate MFs

Here we report on some of the most accurate models that were obtained by the automatic (see above) and manual parameter optimization methods.

- BRISMF#800: manually parameterized MF, with 800 features. Parameter are set to: $K = 800$, $w_{\underline{u}} = -w_{\overline{u}} = w_{\underline{m}} = -w_{\overline{m}} = -0.005, \eta_u = \eta_{ub} = 0.016, \eta_m = \eta_{mb} = 0.005, \lambda_u = \lambda_m = 0.010, \lambda_{ub} = \lambda_{mb} = 0, G = 3.6043$. After 9 epochs, learning rates are multiplied by 0.01, and the model is trained for another 2 epochs.
- SemPosMF#800: this is a semipositive variant of BRISMF#800, where user features are non-negative, item-features are arbitrary. Parameter are set to: $K = 800, w_{\underline{u}} = 0, w_{\overline{u}} = -w_{\underline{m}} = w_{\overline{m}} = 0.005, \eta_u = \eta_{ub} = 0.016, \eta_m = \eta_{mb} = 0.005, \lambda_u = \lambda_m = 0.010, \lambda_{ub} = \lambda_{mb} = 0, G = 3.6043$. After 12 epochs,

Table 3. Probe10 RMSE of accurate MFs without and with applying Q-correction and neighbor corrections (S1 and S2). At column S1 and S2 we also indicated the optimal value of parameter α

#	Model	basic	Q	S1	S2	Q+S1+S2	Combination	basic	Q+S1+S2
1	BRISMF#800	0.8940	0.8930	$0.8916_{(\alpha=8)}$	$0.8914_{(\alpha=7)}$	0.8902			
2	SemPosMF#800	0.8950	0.8941	$0.8916_{(\alpha=8)}$	$0.8913_{(\alpha=5)}$	0.8900	1+2	0.8923	0.8880
3	MlMF#200	0.9112	0.9106	$0.9087_{(\alpha=8)}$	$0.9085_{(\alpha=6)}$	0.9076	1+2+3	0.8913	0.8872
4	MlMF#80	0.9251	0.9240	$0.9104_{(\alpha=9)}$	$0.9072_{(\alpha=2)}$	0.9058	1+2+3+4	0.8909	0.8863
5	BRISMF#1000UM	0.8921	0.8918	$0.8905_{(\alpha=7)}$	$0.8907_{(\alpha=5)}$	0.8901	1+2+3+4+5	0.8895	0.8851
6	MomentumMF	0.9031	0.9020	$0.8979_{(\alpha=6)}$	$0.8956_{(\alpha=3)}$	0.8949	1+2+3+4+5+6	0.8889	0.8838

learning rates are multiplied by 0.01, and the model is trained for another 2 epochs.

- MlMF#200: a BRISMF with 200 features. Automatic parameter optimization.
- MlMF#80: a BRISMF with 80 features. Automatic parameter optimization.
- MomentumMF: a BRISMF with momentum method and 50 features, manually optimized: $K = 50$, $\eta = 0.01$, $\sigma = 0.3$ and $\lambda = 0.00005$. Learnt in 5 epoch.

We refer to a variant of the transductive MF algorithm as Q-correction: in eq. (19) to improve predictions we use only the ratings in the Qualify set, not in the Probe10+Qualify set. See Table 3 for the RMSE values of the each method and their blended versions. We applied two neighborhood corrections on the MF models, with similarities S1 and S2.

The Q, S1, S2 and Q+S1+S2 columns represents results of linear regression of multiple columns (2, 2, 2 and 4 respectively) on Probe10 data. The "basic" column of combinations (1+2, 1+2+3, …, 1+2+3+4+5+6) are combinations of 2, 3, …, 6 columns, resp. The Q+S1+S2 column of combinations (1+2, 1+2+3, …, 1+2+3+4+5+6) are combinations of 8, 12, 16, 20 and 24 columns, resp. In our experiments, blending means ridge regression against the expected Probe10 values.

One can observe in Table 3 that NB correction improves significantly the result of MF based methods. Starting from an average MF (MlMF#80) the reduction of RMSE can be 0.0179, it can reduce the RMSE of the good Momentum MF by 0.0075, and it even improves slightly (0.0026) the very accurate BRISMF#800. In comparison, BellKor's approach ([2], Table 2) results in 0.0096 RMSE reduction, starting from MF with 0.9167 RMSE, here the reduced RMSE score is almost identical with our NB corrected MlMF#80 that has originally only RMSE 0.9251.

If we put in all MFs and all corrections, then the combination yields an RMSE $= 0.8838$. However, it brings only insignificant improvements if one applies Q-correction technique for all MFs. We get RMSE $= 0.8839$ if we exclude the Q-corrections of all MFs but the first from the combination. Moreover, if we apply neighbor and Q-correction only on the first MF, then the RMSE increases only to 0.8850. In general, we can state that one "correction technique" brings major decrease in the RMSE when applied only to a single method in the linear combination. If we apply it multiple times, the improvement becomes less. In other words, Q-corrections or neighbor corrections captures the same aspects of the data, regardless of the MF behind them.

4.2.6 Speed vs. accuracy

It is interesting and important to investigate the relation of accuracy and speed. We run many randomly parameterized MFs with $K = 40$, and collected the best accuracies in each epoch. Table 4 summarizes the results. A Probe10 RMSE of 0.9071 can be achieved within 200 seconds (including the time to train with the 100 million available ratings and evaluate on the Probe10)! Netflix's Cinematch algorithm can achieve Quiz RMSE $= 0.9514$.

Table 4. Probe10 RMSE of fast and accurate MFs.

Epoch	Training Time (sec)	RMSE
1	120	0.9188
2	200	0.9071
3	280	0.9057
4	360	0.9028
5	440	0.9008
6	520	0.9002

To our best knowledge, the running times presented here are far more advantageous than of any other methods published in the literature of CF—though authors usually do not tend to publish running time data. This property pairs with very accurate models which makes the presented models and their implementation to be the most favorable matrix factorization approaches.

4.3 RMSE values reported by other authors

We compare the presented Probe10 RMSE values (which differ from Quiz RMSE values at most by 0.0003) with other RMSE values reported for the Netflix Prize dataset.

Authors often report on Probe RMSE or Quiz RMSE values. Probe RMSE values are calculated by leaving out the Probe set from the Train set, while Quiz RMSE is often computed by incorporating the Probe data into the training of the predictor. Thus, Probe RMSE is often much lower than Quiz RMSE.

Paterek in [10] introduces the use of bias features. He reports on Quiz RMSE $= 0.9070$ for his biased regularized MF (called there RSVD2). Salakhutdinov and Mnih present a Probabilistic MF approach and its variants in [12]. The best result they publish is Quiz RMSE $= 0.8970$, which is obtained as a combination of 3 MFs. Bell and Koren in [2] provides a detailed description of their alternating least squares approach proposed to matrix factorization. Their matrix factorization approach uses a specialized quadratic optimizer instead of ridge regression to assure non-negative weights. They report on Probe RMSE $= 0.9167$ for their MF, and their methods need to run either ridge regression or a specialized quadratic optimizer for $(|I|+|J|)/2 \cdot n^*$ times, on $|\mathcal{R}| \cdot n^*$ data with K input variables in total, where n^* is the number of epochs, which is "few tens". Thus, their method requires $\Omega((|I|+|J|) \cdot K^3/2 + |\mathcal{R}| \cdot K^2) \cdot n^*$ steps using either ridge regression or the specialized quadratic optimizer. They report on Quiz RMSE $= 0.8982$ for their neighbor method executed on the residuals of their MF.

Our presented approaches have $\mathrm{O}(|\mathcal{R}| \cdot K) \cdot n^*$ computational complexity, where n^* is typically less than 20. Remark that $\mathrm{O}(\cdot)$ is an upper bound, while $\Omega(\cdot)$ is a lower bound for computational complexity.

Here we neglected the cost of parameter optimization. Our MF has 13 parameters. We perform the parameter optimization process (Sec. 4.1) with a subset of users (1/6), and with a small K value (typically K is 20 or 40). The optimization process requires 100–200 MF runs. In case of SemPosMF#800, which is manually parameterized, we performed ~ 50 runs. One may argue that parameter optimization for alternating least squares type MF is faster, since there are no learning rates, thus it has just 9 parameters. We observed that the more parameters the MF have, the easier to tune the parameters to get the same Probe10 RMSE. Consequently, we introduced many parameters, for example $\eta_u, \eta_m, \eta_{ub}, \eta_{mb}$ instead of a single η.

The best results in the NP literature are reported in [3], where the authors briefly describe their approach for the Netflix Progress Prize 2007. They use the MF algorithm presented in [2]. They report only on Quiz RMSE values. Here we summarize the best results of that paper:

- Best stand-alone positive MF: Quiz RMSE $= 0.8998$
- Best stand-alone positive MF with "adaptive user factors": Quiz RMSE $= 0.8955$
- Best neighbor method on positive MF: Quiz RMSE $= 0.8953$

Table 5 compares our best methods with the results reported by [3].

Table 5. Comparison of our best Probe10 and Quiz RMSE values against the Quiz RMSE values reported by [3]

Method	Name of our method	Our Probe10	Our Quiz	Bell et al's Quiz
Simple MF	BRISMF #1000	0.8938	0.8939	0.8998
Tweaked MF	BRISMF #1000UM	0.8921	0.8918	N/A
MF with neighbor correction	BRISMF #1000UM, S1	0.8905	0.8904	0.8953

Clearly, our matrix factorization methods and our hybrid MF-neighbor approaches outperform Bell et al's methods: they are more accurate, much faster (in the $O()$ sense as well), simpler, and can handle the change of users' taste in time.

5 Conclusions

This paper presented matrix factorization based approaches for rating based collaborative filtering problems. We proposed a few novel MF variants including a fast (semi-)positive MF, a momentum based MF, a transductive MF, and an incremental variant of MF. The neighbor based correction of MF with two similarity functions was also presented. We reported on the RMSE scores of our algorithms evaluated on the Netflix Prize dataset. We have shown that the presented models outperform the already published ones significantly. It has been also pointed out that our algorithms are very effective in terms of time requirement, needing in general only a few minutes to train reasonably accurate models.

References

[1] R. Bell, Y. Koren, and C. Volinsky. Chasing \$1,000,000: How we won the netflix progress prize. *ASA Statistical and Computing Graphics Newsletter*, 18(2):4–12, December 2007.

[2] R. M. Bell and Y. Koren. Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights. In *Proc of. ICDM, IEEE International Conference on Data Mining*, 2007.

[3] R. M. Bell, Y. Koren, and C. Volinsky. The BellKor solution to the Netflix Prize. Technical report, AT&T Labs Research, 2007. `http://www.netflixprize.com/assets/ProgressPrize2007_KorBell.pdf`.

[4] J. Bennett, C. Eklan, B. Liu, P. Smyth, and D. Tikk. KDD Cup and Workshop 2007. *ACM SIGKDD Explorations Newsletter*, 9(2):51–52, 2007.

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

[6] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proc. of UAI'98, 14th Conference on Uncertainty in Artificial Intelligence*, pages 43–52. Morgan-Kaufmann, 1998.

[7] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[8] T. Hofmann. Latent semantic models for collaborative filtering. *ACM Trans. Inf. Syst.*, 22(1):89–115, 2004.

[9] M. Kurucz, A. A. Benczúr, and K. Csalogány. Methods for large scale svd with missing values. In *Proc. of KDD Cup Workshop at SIGKDD'07, 13th ACM Int. Conf. on Knowledge Discovery and Data Mining*, pages 7–14, San Jose, CA, USA, 2007.

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

[11] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. GroupLens: An open architecture for collaborative filtering of netnews. In *Proc. of CSCW'94, ACM Conference on Computer Supported Cooperative Work*, pages 175–186, Chapel Hill, North Carolina, United States, 1994. ACM Press.

[12] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In J. C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, Cambridge, MA, 2008. MIT Press.

[13] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted Boltzmann machines for collaborative filtering. In *Proc. of ICML'07, the 24th Int. Conf. on Machine Learning*, pages 791–798, Corvallis, OR, USA, 2007.

[14] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proc. of WWW'01: 10th Int. Conf. on World Wide Web*, pages 285–295, Hong Kong, 2001. ACM Press.

[15] N. Srebro, J. D. M. Rennie, and T. S. Jaakkola. Maximum-margin matrix factorization. *Advances in Neural Information Processing Systems*, 17, 2005.

[16] G. Takács, I. Pilászy, B. Németh, and D. Tikk. On the Gravity recommendation system. In *Proc. of KDD Cup Workshop at SIGKDD'07, 13th ACM Int. Conf. on Knowledge Discovery and Data Mining*, pages 22–30, San Jose, CA, USA, 2007.