Journal of Machine Learning Research 10 (2009) 623-656 Submitted 3/08; Revised 11/08; Published 3/09

Scalable Collaborative Filtering Approaches for Large Recommender Systems

Gábor Takács* GTAKACS@SZE.HU
Dept. of Mathematics and Computer Science
Széchenyi István University
Egyetem tér 1.
Győr, Hungary

István Pilászy* PILA@MIT.BME.HU
Dept. of Measurement and Information Systems
Budapest University of Technology and Economics
Magyar Tudósok krt. 2.
Budapest, Hungary

Bottyán Németh* BOTTYAN@TMIT.BME.HU
Domonkos Tikk* TIKK@TMIT.BME.HU
Dept. of Telecom. and Media Informatics
Budapest University of Technology and Economics
Magyar Tudósok krt. 2.
Budapest, Hungary

Editors: Paolo Frasconi, Kristian Kersting, Hannu Toivonen and Koji Tsuda

Abstract

The collaborative filtering (CF) using known user ratings of items has proved to be effective for predicting user preferences in item selection. This thriving subfield of machine learning became popular in the late 1990s with the spread of online services that use recommender systems, such as Amazon, Yahoo! Music, and Netflix. CF approaches are usually designed to work on very large data sets. Therefore the scalability of the methods is crucial. In this work, we propose various scalable solutions that are validated against the Netflix Prize data set, currently the largest publicly available collection. First, we propose various matrix factorization (MF) based techniques. Second, a neighbor correction method for MF is outlined, which alloys the global perspective of MF and the localized property of neighbor based approaches efficiently. In the experimentation section, we first report on some implementation issues, and we suggest on how parameter optimization can be performed efficiently for MFs. We then show that the proposed scalable approaches compare favorably with existing ones in terms of prediction accuracy and/or required training time. Finally, we report on some experiments performed on MovieLens and Jester data sets.

Keywords: collaborative filtering, recommender systems, matrix factorization, neighbor based correction, Netflix prize

∗. All authors also affiliated with Gravity R&D Ltd., 1092 Budapest, Kinizsi u. 11., Hungary; info@gravityrd.com.

1. Introduction

Recommender systems attempt to profile user preferences over items, and model the relation between users and items. The task of recommender systems is to recommend items that fit a user's tastes, 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 last but not least the price of the items is modest. Typical areas of such services are mostly related to art (esp. books, movies, music), fashion, food and restaurants, gaming and humor.

With the burgeoning of web based businesses, an increasing number of web based merchant or rental services use recommender systems. Some of the major participants of e-commerce web, like Amazon 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 the recommender system of Netflix (see details in Section 5.1.1).

There are two basic strategies that can be applied when generating recommendations. *Content-based approaches* profile users and items by identifying their characteristic features, such us demographic data for user profiling, and product information/descriptions for item profiling. The profiles are used by algorithms to connect user interests and item descriptions when generating recommendations. However, it is usually laborious to collect the necessary information about items, and similarly it is often difficult to motivate users to share their personal data to help create the database for the basis of profiling.

Therefore, the alternative approach, termed *collaborative filtering* (CF), which makes use of only past user activities (for example, transaction history or user satisfaction expressed in ratings), is usually more feasible. CF approaches can be applied to recommender systems independently of the domain. CF algorithms identify relationships between users and items, and make associations using this information to predict user preferences.

In this paper, we focus 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, titles or artifacts, usually on a discrete numerical scale, and the system then recommends other items based on ratings the *virtual community* has already provided. The virtual community was defined by Hill et al. (1995) as "a group of people who share characteristics and interact in essence or effect only." The underlying assumption is that people who had similar tastes in the past may also agree on their tastes in the future.

1.1 Related Work

The first works on the field of CF were published in the early 1990s. Goldberg et al. (1992) presented the Tapestry system that used collaborative filtering to filter mails simultaneously from several mailing lists, based on the opinion of other users on the readings. Resnick et al. (1994) described the GroupLens system that was one of the pioneer applications of the field where users could rate articles on a 1–5 scale after having read them and were then offered suggestions. Breese et al. (1998)

divided the underlying techniques of predicting user preferences into two main groups. *Memory-based* approaches operate on the entire database of ratings collected by the vendor or service supplier. On the other hand, *model-based* approaches use the database to estimate or learn a model and then apply this model for prediction.

Over the last broad decade many CF algorithms have been proposed that approach the problem by different techniques, including similarity/neighborhood based approaches (Resnick et al., 1994; Sarwar et al., 2001), personality diagnosis (Pennock et al., 2000), Bayesian networks (Breese et al., 1998), restricted Boltzmann machines (RBM) (Salakhutdinov et al., 2007), and various matrix factorization techniques (Canny, 2002; Hofmann, 2004; Sarwar et al., 2000; Srebro et al., 2005). Vozalis and Margaritis (2007) presented an MF approach that incorporates demographic information and ratings to enhance plain CF algorithms. Breese et al. (1998) surveyed in detail the major CF approaches of the early years, Rashid et al. (2006) gave a short description of most recent methods, and Adomavicius and Tuzhilin (2005) also investigated in their survey the possible extensions.

The NP competition boosted the interest in CF, and yielded the publication of a number of new methods. We should also mention here the NP related KDD Cup and Workshop (Bennett et al., 2007), which indicated the possible directions of scalable and accurate CF methods. We feature among them the matrix factorization, neighbor based approaches, and their combinations.

Several matrix factorization techniques have been successfully applied to CF, including singular value decomposition (Sarwar et al., 2000), probabilistic latent semantic analysis (Hofmann, 2004), probabilistic matrix factorization (Salakhutdinov and Mnih, 2008), maximum margin matrix factorization (Srebro et al., 2005), expectation maximization for MF (Kurucz et al., 2007), and alternating least squares (Bell and Koren, 2007a).

Simon Funk (Brandyn Webb) published a detailed implementation of a regularized MF with separate feature update.[1] Paterek (2007) introduced a bunch of novel techniques, including MF with biases, applying kernel ridge regression on the residual of MF, linear model for each item, and asymmetric factor models (NSVD1, NSVD2). Kurucz et al. (2007) showed the application of expectation maximization based MF methods for the NP. Bell and Koren (2007a) presented an improved neighborhood based approach, which removes the global effect from the data, and calculates optimal similarity values by solving regression problems. Salakhutdinov et al. (2007) mention without details a momentum based MF method that uses batch learning.

Our methods are different from the above ones in various aspects. Bell and Koren (2007a) use alternate least squares, but we use incremental gradient descent (also known as stochastic gradient descent) method for weight updates. Their method does not use the chronological order of ratings, while we exploit this information in our approaches. They use only positive and ordinary MFs, while we propose the semi-positive version of the MF algorithm. Paterek (2007) applies a different learning scheme following Simon Funk's path (see the detailed comparison in Section 3.1). We point out that this difference makes our training process somewhat faster and does not deteriorate the accuracy. Other differences are that Paterek uses fewer meta-parameters for his MF methods. Salakhutdinov et al. (2007) apply a momentum based MF with batch learning, but we point out that incremental gradient descent is more appropriate for large scale problems.

1. Details at `http://sifter.org/~simon/journal/20061211.html`.

1.2 Main Results and Organization

The aim of this work is to propose accurate and scalable solutions for a class of collaborative filtering problems. Scalability is crucial since CF systems often have to manage millions of users or items. As it was shown by many researchers (see for example Bell et al., 2007b) combination (also termed blending) of various algorithms typically outperforms single methods in terms of accuracy. Therefore, we would like to emphasize here that a particular method can have beneficial impact on the overall accuracy if (1) it yields a very accurate model on its own or (2) it "blends well", that is it improves the accuracy of the combination of several models. The methods proposed in this paper satisfy one or both of these criteria. We classify the methods presented in this paper into 4 categories:

- crucial in improving accuracy or run-time;
- may negligibly improve accuracy, in a way that is more important for competitions than for real life data;
- does not improve accuracy, but is useful for blending with other techniques;
- important for real life data.

This paper is organized as follows. Section 2 defines the CF setting we focus on in this work. Section 3 describes our proposed MF algorithms. In particular, we present

- a regularized MF and its biased version that use an incremental gradient descent weight updating scheme, termed as RISMF and BRISMF, (biased) regularized incremental simultaneous MF;
- a fast (semi-)positive MF version that approximates the features by using nonnegative values for either users or items or both;
- an accurate momentum based MF approach;
- an incremental variant of MF that efficiently handles new users/ratings (this is crucial in a real-life recommender systems);
- 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.

We also introduce a special MF version that supports the visualization of user/item features, which can be used to generate explanation for recommendations. Finally, we illustrate that the learning scheme of some MF algorithms can be directly described in the neural network framework.

Section 4 presents a neighbor based correction approach for MF, which alloys the global perspective of MF and the localized property of neighbor based approaches efficiently. We propose here two similarity functions for items.

Section 5 contains the experiments. First, we introduce the Netflix Prize data set against which our proposed algorithms are validated. In addition, we also describe the MovieLens and the Jester data sets, on which we also run some experiments. Next, we mention some important implementation issues: we comment on the efficient storage of very large rating databases, we consider the proper ordering of training examples, and we outline our parameter optimization heuristic. We then

report on our comprehensive experiments run on the NP data set. In order to illustrate the applicability of MF methods to other data sets, we also report on tests executed with a selected MF method, BRISMF, on the MovieLens and Jester data sets and compare the results with baseline predictors. For the NP data set, we report on the results of each proposed method and the most accurate combinations of methods. We briefly analyze the accuracy-time complexity trade-off of MF methods. We finally compare our results to other published ones to show that the proposed methods are comparable to the existing one in terms of root mean squared error (RMSE). We also discuss the time complexity of some selected methods, and we point out that it compares favorably to similar type methods published so far. We further mention here that all the presented methods are part of the blended solution of our team Gravity in the NP contest.

2. Problem Definition

We define the problem of *collaborative filtering* (CF) in the following setting. The problem can be modeled by the random triplet (U,I,R), where

- U taking values from $\{1,\dots,N\}$ is the *user identifier* (N is the number of users),
- I taking values from $\{1,\dots,M\}$ is the *item identifier* (M is the number of items), and
- R taking values from $\mathcal{X} \subset \mathbb{R}$ is the rating value. Typical rating values can be binary ($\mathcal{X} = \{0,1\}$), integers from a given range (for example, $\mathcal{X} = \{1,2,3,4,5\}$), or real numbers of a closed interval (for example, $\mathcal{X} = [-10,10]$).

A realization of (U,I,R) denoted by (u,i,r) means that user u rated item i with value r.

The goal is to estimate R from (U,I) such that the root mean squared error of the estimate,

$$\text{RMSE} = \sqrt{\mathbf{E}\{(\hat{R}-R)^2\}}, \tag{1}$$

is minimal, where $\hat{R}$ is the estimate[2] of R. We briefly discuss other possible evaluation metrics in CF in Section 5.

In practice, the distribution of (U,I,R) is not known: we are only given a finite sample, $\mathcal{T}' = \{(u_1,i_1,r_1),(u_2,i_2,r_2),\dots,(u_t,i_t,r_t)\}$, generated by it. The sample $\mathcal{T}'$ can be used for training predictors. We assume *sampling without replacement* in the sense that (user ID, item ID) pairs are unique in the sample, which means that users do not rate items more than once. Let us introduce the notation $\mathcal{T} = \{(u,i) : \exists r : (u,i,r) \in \mathcal{T}'\}$ for the set of (user ID, item ID) pairs. Note that $|\mathcal{T}'| = |\mathcal{T}|$, and typically $|\mathcal{T}| \ll N \cdot M$, because most of the users rate only a small subset of the entire set of items. The sample can be represented as a partially specified matrix denoted by $\mathbf{R} \in \mathbb{R}^{N \times M}$, where the matrix elements are known in positions $(u,i) \in \mathcal{T}$, and unknown in positions $(u,i) \notin \mathcal{T}$. The value of the matrix $\mathbf{R}$ at position $(u,i) \in \mathcal{T}$, denoted by r_{ui}, stores the rating of user u for item i. For clarity, we use the term (u,i)-th rating in general for r_{ui}, and (u,i)-th training example if $r_{ui} : (u,i) \in \mathcal{T}$.

2. In general, superscript "hat" denotes the prediction of the given quantity, so $\hat{x}$ is the prediction of x.

The goal of this CF setup is to create such predictors that aim at minimizing the error (1). In practice, we cannot measure the error because the distribution of (U,I,R) is unknown, but we can estimate the error on a validation set. Let us denote the validation set by $\mathcal{V}' \subset [1,\ldots,N] \times [1,\ldots,M] \times \mathcal{X}$, assuming sampling without replacement as defined above, and we further assume the uniqueness of (user ID, item ID) pairs across $\mathcal{T}'$ and $\mathcal{V}'$. We define $\mathcal{V} = \{(u,i) : \exists r : (u,i,r) \in \mathcal{V}'\}$. The assumptions ensure that $\mathcal{T} \cap \mathcal{V} = \emptyset$. If both the training set $\mathcal{T}'$ and validation set $\mathcal{V}'$ are generated from the same distribution the estimate of RMSE can be calculated as

$$\widehat{\mathrm{RMSE}} = \sqrt{\frac{1}{|\mathcal{V}|} \sum_{(u,i)\in\mathcal{V}} (\hat{r}_{ui} - r_{ui})^2}.$$

For better readability, from now on we omit the "hat" from the RMSE, recalling that we always calculate the estimate of the error.

When we predict a given rating r_{ui} by $\hat{r}_{ui}$ we refer to the user u as *active user* and to the item i as *active item*. The (u,i) pair of active user and active item is termed *query*.

3. Matrix Factorization

Matrix factorization is one of the most often applied techniques for CF problems. Numerous different MF variants have been already published and were validated against the NP data set as well. We should credit here again Simon Funk, who published the first detailed implementation notes on this problem. He applied a regularized MF with gradient descent learning scheme. His model trained factors one after another, which can be considered as a series of 1 factor MF training processes performed on the residual of the previous one. Other efficient MF variants were published by Bell and Koren (2007a), where they used alternating least squares for weight updates. Biased MF was applied by Paterek (2007), whose technique was also proposed at the same time in our previous work (Takács et al., 2007) as "constant values in matrices".

In this section we give an overview of our MF variants that proved to be effective in tackling the large scale practical problem of NP, and hence may be considered as a useful collection of tools for practitioners. Here we also propose several modifications for already known MF variants, which are effective in improving the accuracy of the generated models. We also give details concerning the time requirement of certain methods and propose efficient implementation solutions.

The idea behind MF techniques is very simple. Suppose we want to approximate the matrix $\mathbf{R}$ as the product of two matrices:

$$\mathbf{R} \approx \mathbf{PQ},$$

where $\mathbf{P}$ is an $N \times K$ and $\mathbf{Q}$ is a $K \times M$ matrix. We call $\mathbf{P}$ the user feature matrix and $\mathbf{Q}$ the item feature matrix, and K is the number of features in the given factorization. If we consider the matrices as linear transformations, the approximation can be interpreted as follows: matrix $\mathbf{Q}$ is a transform from $\mathcal{S}_1 = \mathbb{R}^M$ into $\mathcal{S}_2 = \mathbb{R}^K$, and matrix $\mathbf{P}$ is a transform from $\mathcal{S}_2$ into $\mathcal{S}_3 = \mathbb{R}^N$. Typically, $K \ll N$ and $K \ll M$, therefore the vector space $\mathcal{S}_2$ acts as a bottleneck when predicting $\mathbf{v}_3 \in \mathcal{S}_3$ from $\mathbf{v}_1 \in \mathcal{S}_1$. In other words, the number of parameters to describe $\mathbf{R}$ can be reduced from $|\mathcal{T}|$ to $NK+KM$. Note that $\mathbf{Q}$ and $\mathbf{P}$ typically contain real numbers, even when $\mathbf{R}$ contains only integers.

In the case of the given problem, the unknown ratings of $\mathbf{R}$ cannot be represented by zero. For this case, the approximation task can be defined as follows. Let p_{uk} denote the elements of

$\mathbf{P} \in \mathbb{R}^{N \times K}$, and q_{ki} the elements of $\mathbf{Q} \in \mathbb{R}^{K \times M}$. Further, let $\mathbf{p}_u$ denote a row (vector) of $\mathbf{P}$, and $\mathbf{q}_i$ a column (vector) of $\mathbf{Q}$. Then:

$$\hat{r}_{ui} = \sum_{k=1}^{K} p_{uk} q_{ki} = \mathbf{p}_u \mathbf{q}_i, \tag{2}$$

$$e_{ui} = r_{ui} - \hat{r}_{ui} \quad \text{for } (u,i) \in \mathcal{T},$$

$$e'_{ui} = \frac{1}{2} e_{ui}^2, \tag{3}$$

$$\text{SSE} = \sum_{(u,i) \in \mathcal{T}} e_{ui}^2 = \sum_{(u,i) \in \mathcal{T}} \left(r_{ui} - \sum_{k=1}^{K} p_{uk} q_{ki} \right)^2,$$

$$\text{SSE}' = \frac{1}{2} \text{SSE} = \sum_{(u,i) \in \mathcal{T}} e'_{ui},$$

$$\text{RMSE} = \sqrt{\text{SSE}/|\mathcal{T}|},$$

$$(\mathbf{P}^*, \mathbf{Q}^*) = \arg\min_{(\mathbf{P},\mathbf{Q})} \text{SSE}' = \arg\min_{(\mathbf{P},\mathbf{Q})} \text{SSE} = \arg\min_{(\mathbf{P},\mathbf{Q})} \text{RMSE}. \tag{4}$$

Here $\hat{r}_{ui}$ denotes how the u-th user would rate the i-th item, according to the model, e_{ui} denotes the training error measured at the (u,i)-th rating, and SSE denotes the sum of squared training errors. Eq. (4) states that the optimal $\mathbf{P}$ and $\mathbf{Q}$ minimize the sum of squared errors only on the known elements of $\mathbf{R}$.

In order to minimize RMSE, which is in this case equivalent to minimizing SSE$'$, we apply a simple incremental gradient descent method to find a local minimum of SSE$'$, where one gradient step intends to decrease the square of prediction error of only one rating, or equivalently, either e'_{ui} or e_{ui}^2.

Minimizing RMSE can be seen as a weighted low-rank approximation of $\mathbf{R}$. Weighted low-rank approximations try to minimize the objective function $\text{SSE}_\mathbf{W} = \sum_{u=1}^{N} \sum_{i=1}^{M} w_{ui} \cdot e_{ui}^2$, where w_{ui}-s are predefined non-negative weights. For collaborative filtering problems, w_{ui} is 1 for known ratings, and 0 for unknown ratings. Srebro and Jaakkola (2003) showed that when the rank of $((w_{ui}))$ is 1, all local minima are global minima. However, when it is greater than 1—as in the case of collaborative filtering—this statement does not hold any more, which was shown by the authors via counterexamples.

For the incremental gradient descent method, suppose we are at the (u,i)-th training example, r_{ui}, and its approximation $\hat{r}_{ui}$ is given.

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

$$\frac{\partial}{\partial p_{uk}} e'_{ui} = -e_{ui} \cdot q_{ki}, \qquad \frac{\partial}{\partial q_{ki}} e'_{ui} = -e_{ui} \cdot p_{uk}.$$

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

$$p'_{uk} = p_{uk} + \eta \cdot e_{ui} \cdot q_{ki},$$
$$q'_{ki} = q_{ki} + \eta \cdot e_{ui} \cdot p_{uk}.$$

That is, we change the weights in $\mathbf{P}$ and $\mathbf{Q}$ to decrease the square of actual error, thus better approximating r_{ui}. Here η is the learning rate. This basic MF method is referred to as ISMF, that is

incremental simultaneous MF, due to its distinctive incremental and simultaneous weight updating method to other MF methods.

When the training has been finished, each value of $\mathbf{R}$ can be computed easily using Eq. (2), even at unknown positions. In other words, the model ($\mathbf{P}^*$ and $\mathbf{Q}^*$) provides a description of how an arbitrary user would rate any item.

3.1 RISMF

The matrix factorization presented in the previous section can overfit for users with few (no more than K) ratings: assuming that the feature vectors of the items rated by the user are linearly independent and $\mathbf{Q}$ does not change, there exists a user feature vector with zero training error. Thus, there is a potential for overfitting, if η and the extent of the change in $\mathbf{Q}$ are both small. A common way to avoid overfitting is to apply regularization by penalizing the square of the Euclidean norm of weights. This is often used in machine learning methods, for example Support Vector Machines and ridge regression apply that. It is common also in neural networks, where it is termed as weight decay (Duda et al.). Penalizing the weights results in a new optimization problem:

$$\begin{aligned} e'_{ui} &= (e_{ui}^2 + \lambda \cdot \mathbf{p}_u \cdot \mathbf{p}_u^T + \lambda \cdot \mathbf{q}_i^T \cdot \mathbf{q}_i)/2, \\ \mathrm{SSE}' &= \sum_{(u,i)\in\mathcal{T}} e'_{ui}, \\ (\mathbf{P}^*, \mathbf{Q}^*) &= \arg\min_{(\mathbf{P},\mathbf{Q})} \mathrm{SSE}'. \end{aligned} \tag{5}$$

Here $\lambda \geq 0$ is the regularization factor. Note that minimizing SSE$'$ is no longer equivalent to minimizing SSE, unless $\lambda = 0$, in which case we get back the ISMF. We call this MF variant RISMF that stands for regularized incremental simultaneous MF.

Similar to the ISMF approach, we compute the gradient of e'_{ui}:

$$\frac{\partial}{\partial p_{uk}} e'_{ui} = -e_{ui} \cdot q_{ki} + \lambda \cdot p_{uk}, \qquad \frac{\partial}{\partial q_{ki}} e'_{ui} = -e_{ui} \cdot p_{uk} + \lambda \cdot q_{ki}. \tag{6}$$

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

$$\begin{aligned} p'_{uk} &= p_{uk} + \eta \cdot (e_{ui} \cdot q_{ki} - \lambda \cdot p_{uk}), \\ q'_{ki} &= q_{ki} + \eta \cdot (e_{ui} \cdot p_{uk} - \lambda \cdot q_{ki}). \end{aligned} \tag{7}$$

For the training algorithm used in RISMF, see Algorithm 1. Note that we use early stopping in Algorithm 1, thus $\mathbf{P}^*$ and $\mathbf{Q}^*$ differs from Eq. (5), because we optimize for the validation set. Note that the matrices are initialized randomly. If both $\mathbf{P}$ and $\mathbf{Q}$ are initialized with a constant value, that is, both are rank 1, the weight update will not increase the rank, which is equivalent to the $K = 1$ case. Random initialization is a simple way to avoid this. Typically, we uniformly choose random values from $[-0.01, 0.01]$ or $[-0.02, 0.02]$.

We point out that RISMF differs from Funk's MF in a few important aspects. We update each feature simultaneously and initialize the matrix randomly. Simon Funk's approach learns each feature separately during a certain number of epochs, but there are no specification as to when the learning procedure has to step to the next feature. His approach converges slower than ours, because it iterates over $\mathbf{R}$ more times. Both methods use regularization and early stopping to prevent overfitting.

Input: $\mathcal{T}'$: training set, η: learning rate, λ: regularization factor
Output: $\mathbf{P}^*$,$\mathbf{Q}^*$: the user and item feature matrices
Partition $\mathcal{T}'$ into two sets: ${\mathcal{T}'}_{\mathrm{I}}$, ${\mathcal{T}'}_{\mathrm{II}}$ (validation set)
Initialize **P** and **Q** with small random numbers.
loop until the terminal condition is met. One epoch:
    iterate over each (u,i,r_{ui}) element of ${\mathcal{T}'}_{\mathrm{I}}$:
        compute e'_{ui};
        compute the gradient of e'_{ui}, according to Eq. (6);
        for each k
            update $\mathbf{p}_u$, the u-th row of **P**,
            and $\mathbf{q}_i$, the i-th column of **Q** according to Eq. (7);
    calculate the RMSE on ${\mathcal{T}'}_{\mathrm{II}}$;
    if the RMSE on ${\mathcal{T}'}_{\mathrm{II}}$ was better than in any previous epoch:
        Let $\mathbf{P}^* = \mathbf{P}$ and $\mathbf{Q}^* = \mathbf{Q}$.
    terminal condition: RMSE on ${\mathcal{T}'}_{\mathrm{II}}$ does not decrease during two epochs.
end

Algorithm 1: Training algorithm for RISMF

We observed (Takács et al., 2007), that the learning curve of epochs (RMSE on the validation set as a function of the number of epochs) is always convex, regardless of the value of λ, that is why we use not only regularization but also early stopping.

3.2 BRISMF: Constant Values in Matrices

One may argue that some users tend to rate all items higher or lower than the average. The same may hold for items: some items can be very popular. Although MF can reconstruct the original matrix exactly when K is large enough and $\lambda = 0$, this is not the case when overfitting is to be avoided. There is a straightforward way to extend RISMF to be able to directly model this phenomenon, by extending MF with biases for users and items.

The bias feature idea was mentioned by Paterek (2007). He termed his version RSVD2, which appeared at the same time in our work in a generalized form by incorporating constant values in the MF (Takács et al., 2007). Paterek's and our variants share some common features, but he used Simon Funk's approach to update feature weights.

We incorporate bias features into RISMF by fixing the first column of **P** and the second row of **Q** to the constant value of 1. By "fixing to a constant value" we mean initialize $\mathbf{p}_{\bullet 1}$ and $\mathbf{q}_{2\bullet}$ with a fixed constant value instead of random values and drop the application of (7) when updating $\mathbf{p}_{\bullet 1}$ and $\mathbf{q}_{2\bullet}$. In this way, we get back exactly Paterek's RSVD2, except that we update features simultanously. The pair of these features ($\mathbf{q}_{1\bullet}$ and $\mathbf{p}_{\bullet 2}$) can serve as bias features. Our $\mathbf{p}_{u2}$ and $\mathbf{q}_{1i}$ corresponds to Paterek's c_i and d_j resp.

We refer to this method as BRISMF that stands for biased regularized incremental simultaneous MF. This simple extension speeds up the training phase and yields a more accurate model with better generalization performance. Since BRISMF is always superior to RISMF in terms of both the accuracy and the running time, it is our recommended basic MF method.

3.3 Semipositive and Positive MF

The RISMF algorithm can generate not only positive but also negative feature values. Nonnegative matrix factorization (Lee and Seung, 1999) generates models with only nonnegative features, which enables additive part-based representation of the data. Positive (Bell and Koren, 2007a) and semipositive (Hofmann, 2004) matrix factorization techniques have been successfully applied in the field of CF. We present a simple modification of RISMF that can give positive and semipositive factorizations. Although, these modifications do not yield more accurate models, they are important, because the models are still accurate (Section 5.4.6), they blend well with other methods (Section 5.4.8), and the running time (Section 5.4.10), simplicity and accuracy compares favorably with Bell and Koren's positive MF method.

We talk about semipositive MF when exactly one of **P** and **Q** contains both nonnegative and negative values, and positive MF, when both contains only nonnegative values.

We apply a simple thresholding method to ensure the nonnegativeness of features: for the (u,i)-th training example in a given epoch, if p_{uk} or q_{ki} would become negative when applying (7), we reset their value to 0. We describe the modified equations for the case when both user and item features are required to be nonnegative:

$$\begin{aligned} p'_{uk} &= \max\{0, p_{uk} + \eta \cdot e_{ui} \cdot q_{ki} - \lambda \cdot p_{uk}\}, \\ q'_{ki} &= \max\{0, q_{ki} + \eta \cdot e_{ui} \cdot p_{uk} - \lambda \cdot q_{ki}\}. \end{aligned} \tag{8}$$

If we allow, for instance, user features to be negative, we can simply use Eq. (7) instead of Eq. (8) for p'_{uk}. Allowing only nonnegative item features can be treated similarly.

3.4 Applying Momentum Method

This method modifies the learning rule of RISMF slightly. In each learning step the weight updates are calculated from the actual gradient and from the last weight changes. With the modification of (7) we get the following equations:

$$\begin{aligned} p_{uk}(t+1) &= p_{uk}(t) + \Delta p_{uk}(t+1), \\ \Delta p_{uk}(t+1) &= \eta \cdot (e_{ui} \cdot q_{ki}(t) - \lambda \cdot p_{uk}(t)) + \sigma \cdot \Delta p_{uk}(t), \\ q_{ki}(t+1) &= q_{ki}(t) + \Delta q_{ki}(t+1), \\ \Delta q_{ki}(t+1) &= \eta \cdot (e_{ui} \cdot p_{uk}(t) - \lambda \cdot q_{ki}(t)) + \sigma \cdot \Delta q_{ki}(t). \end{aligned}$$

Here σ is the momentum factor and $p_{uk}(t+1)$ and $p_{uk}(t)$ stands for the new and old k-th feature values of user u, respectively. Analogously, $\Delta p_{uk}(t+1)$ and $\Delta p_{uk}(t)$ denote the current and last change of the given feature value. The notations are similar for the item features.

Without a detailed description, the momentum method was mentioned by Salakhutdinov et al. (2007), but they used batch learning, which makes the training slower. In our experiments, momentum MF does not yield more accurate models; however, it blends well with other methods (Section 5.4.8).

3.5 Retraining User Features

The incremental gradient descent weight update of RISMF has a serious drawback: item features change while we iterate through users. If the change is significant, user features updated in the

beginning of an epoch may be inappropriate at the end of the epoch. We found two ways to solve this.

1. We can evaluate the test ratings of a user immediately after iterating through its ratings in the training set, and before starting to iterate through the next user's ratings.
2. We can completely recompute the user features after the learning procedure. This method can serve as an efficient incremental training method for recommender systems. Algorithm 2 summarizes the method.

Input: $\mathcal{T}'$: training set, η: learning rate, λ: regularization factor
Output: $\mathbf{P}^*$,$\mathbf{Q}^*$: the user and item feature matrices
1 Partition $\mathcal{T}'$ into two sets: $\mathcal{T}'_{\mathrm{I}}$, $\mathcal{T}'_{\mathrm{II}}$ (tuning set)
2 First training step: call Algorithm 1, store the result in $(\mathbf{P}_1, \mathbf{Q}_1)$
3 Let $\mathbf{Q} = \mathbf{Q}_1$, initialize $\mathbf{P}$ randomly.
4 Second training step: call Algorithm 1, with the following restrictions:
5 skip the weight initialization step, use $\mathbf{P}$ and $\mathbf{Q}$ from here.
6 do not change weights in $\mathbf{Q}$, that is, do not apply (7) or its variants for q_{ki}.
7 store the optimal number of epochs, denote it n^*.
8 Return the result of Algorithm 1 called in line 4
9 **end**

Algorithm 2: Algorithm for retraining user features

Note that this method can efficiently incorporate into the model new users or new ratings of existing users without the necessity of retraining the entire model, which is very important for recommender systems. Then we do not apply the whole training procedure, just reset the user feature weights of the active user u and apply the second training procedure for n^* epochs for u. The second training procedure needs to iterate through the entire database—which requires slow (but sequential) disk access operation—only once (not n^* times), as the ratings of user u can be kept in memory and can be immediately re-used in the next epoch.

We remark that the presented algorithm cannot handle the addition of new items, and after the addition of many new ratings to the database, $\mathbf{Q}$ will be obsolete, thus the first training step should be re-run. Retraining user features mostly yields a slightly more accurate model (Section 5.4.7), which means that it is useful for real life recommender systems to handle the addition of new users or ratings.

3.6 Transductive MF

Transductive learning involves the use of validation examples but not their labels. In the case of CF, this means that the algorithm "knows" what validation examples the model will be applied for, the $(u,i) \in \mathcal{V}$ pairs, but not the corresponding r_{ui} values. The first transductive model for CF was the conditional restricted Boltzmann machine (Salakhutdinov et al., 2007). In this RBM variant, the distribution over the visible and hidden units is defined conditional on which items the user has rated. The authors noted that the model performance is significantly improved by using conditional distribution instead of unconditional one.

We give a possible practical example when transductive learning model is useful. Let us suppose that, when buying items, users can optionally rate them. Via the proposed technique, the information

that a user purchased but did not rate an item can be incorporated in such cases into the prediction model.

We propose a transductive MF that can exploit this information after the learning process. The idea behind the method is the following: suppose that user u (with feature vector $\mathbf{p}_u$) has the following v queries in the validation set: $r_{u1},\ldots,r_{uv}$. When we are at the i-th validation example of user u, we can predict another feature vector based only on what other items ($1 \leq i' \leq v, i' \neq i$) are to be predicted. A proper linear combination—not depending on u or i—of the original user feature vector and this new feature vector can yield a third feature vector for the prediction of r_{ui} that produces a better predictor than the original one (see Table 7 for the performance gain obtained by transductive MF). Formally:

$$\mathbf{p}'_u(j) = \frac{1}{\sqrt{|i' : (u,i') \in \mathcal{V}| + 1}} \sum_{\substack{i''=1 \\ i'' \neq i}}^{v} \mathbf{q}_{i''}^T. \tag{9}$$

$$\hat{r}'_{ui} = \hat{r}_{ui} + \nu \cdot \mathbf{p}'_u(j) \cdot \mathbf{q}_i = \mathbf{p}_u \cdot \mathbf{q}_i + \nu \cdot \mathbf{p}'_u(j) \cdot \mathbf{q}_i.$$

The attenuation factor in Eq. (9) ensures that the more ratings a user has in the training set, the less the prediction relies on the information the validation set provides, thus $\hat{r}'_{ui}$ will differ less from $\hat{r}_{ui}$.

In practice, ν need not be determined: we can use $\hat{r}'_{ui}$ and $\mathbf{p}'_u(j) \cdot \mathbf{q}_i$ as two predictions for r_{ui}, and apply linear regression to get an improved RMSE. Transductive MF is a post-processing method for MF, which can exploit the information provided by the existence of ratings even if the values of the ratings are unknown. This can be the case in some real life problems, for example, when a user buys many items but rates only a few of them.

3.7 2D Variant of the Matrix Factorization Algorithm

We propose here an MF variant that provides visual information about MF features, which can be used to generate an explanation for recommendations. The idea is that we store user and item features in a two dimensional (2D) grid instead of a one dimensional row vector. Accordingly, we replace the p_{uk} and q_{ki} notation of feature values with p_{ukl} and q_{imn}. Furthermore we define a simple neighborhood relation between the features based on their distance in the grid. If the difference of the horizontal and vertical positions of two features is small, then the "meaning" of those features should also be close. To achieve this, we modify our error function in (3) to penalize the difference between neighbor features:

$$e_{ui}{}'' = e'_{ui} + \sum_{k,l} \sum_{m \neq k, n \neq l} s(k,l,m,n)(p_{ukl} - p_{umn})^2 + \sum_{k,l} \sum_{m \neq k, n \neq l} s(k,l,m,n)(q_{ikl} - q_{imn})^2.$$

Here (k,l), (m,n) are the indices in the 2D grid, and $s(k,l,m,n)$ is the similarity between the (k,l)-th and (m,n)-th positions of the grid. For example we can use the inverse of the squared Euclidean distance as similarity:

$$s(k,l,m,n) = \frac{\rho}{(k-m)^2 + (l-n)^2}.$$

With this function the gradient used for learning p_{ukl} and q_{ikl} will be:

$$\frac{\partial}{\partial p_{ukl}} e_{ui}'' = \frac{\partial}{\partial p_{ukl}} e_{ui}' + 2 \sum_{m \neq k, n \neq l} s(k,l,m,n)(p_{ukl} - p_{umn}),$$

$$\frac{\partial}{\partial q_{ikl}} e_{ui}'' = \frac{\partial}{\partial q_{ikl}} e_{ui}' + 2 \sum_{m \neq k, n \neq l} s(k,l,m,n)(q_{ikl} - q_{imn}).$$

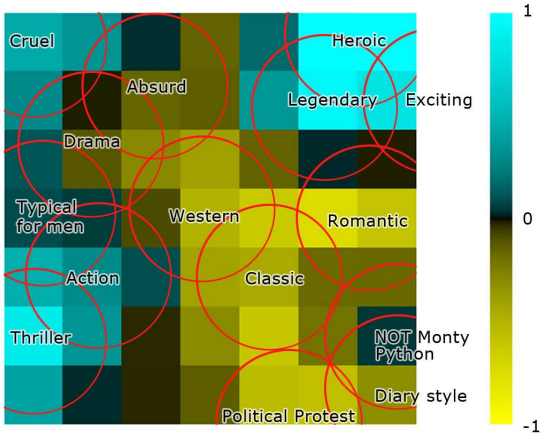

Figure 1: Features of movie *Constantine*

The 2D features of items (or users) can be visualized on a so-called *feature map*. In our next examples, items are movies from the Netflix Prize data set. Meanings can be associated to the regions of the feature map. Note that the labels on Figure 1 are assigned to *groups of features* and not to single features. The labels have been manually determined based on such movies that have extreme values at the given feature group.

The labeling process is performed as follows: we select a cell of the feature map. Then we list the movies with the highest and the lowest feature values in the selected cell; we term them *representative movies*. Finally, we select an expression which describes the best the common properties of the movies with high feature values in contrast to the movies with low feature values. In the case when neighboring features have similar representative movies, we assign a common label to the corresponding cells. As a result of the label unification, we can obtain larger areas of the feature map with the same label; see for instance labels *Classic*, *Western* or *Action* on Figure 1. Since the described labeling process is subjective, it requires human interaction. The size of the areas the labels are assigned to depends on multiple factors like the neighborhood strength ρ, the size of the feature map, and how general the label term assigned is. In the example (Figure 1), the labels principally characterize the features directly under themselves and immediate neighbors. The further a feature is from the label, the less the label reflects its meaning.

Such feature maps are useful for detecting main differences between movies or episodes of the same movie. Figure 2 represents the three episodes of The Matrix movie. One can observe that the main characteristic of the feature maps are the same, but there are noticeable changes between the first and the other episodes. In the first episode the feature values are higher in the area of *Political protest* and *Absurd* and lower around *Legendary*. This may indicate that the first episode presents a unique view of the world, but the other two are rather based on the success of the first

episode. Visual feature maps can also be used to demonstrate to the users why they are provided the current recommendations: showing similar feature map of items formerly ranked high by the user can justify the recommendations. This kind visual of explanation may be a preferred by people who can capture the meaning of visual information faster than textual one. We, therefore, recommend this technique for real life recommender systems, although it does not improve the blending of the methods.

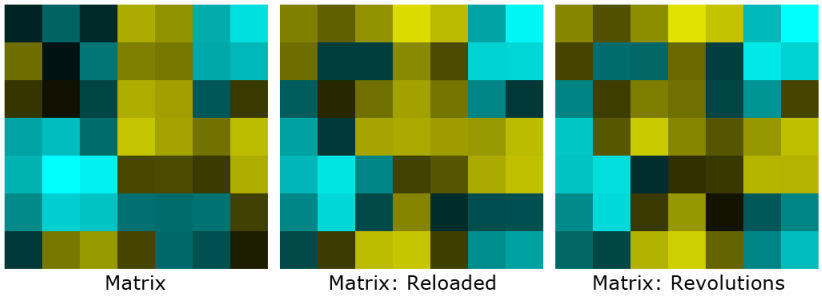

Figure 2: Features of *The Matrix* episodes

3.8 Connections with Neural Networks

In this section we point out that the learning scheme of some MF variants can be directly represented in the framework of neural networks (NN). We show that this correspondence can be exploited by applying the methodology of NN learning for CF problems.

The learning of the ISMF and RISMF models can be paired with the multi-layer perceptron depicted on Figure 3. The network has N inputs, M outputs and K hidden neurons and an identity activation function in each neuron. The weight between the u-th input and the k-th hidden neuron corresponds to p_{uk}, and the weight between the k-th hidden neuron and the i-th output neuron corresponds to q_{ki}.

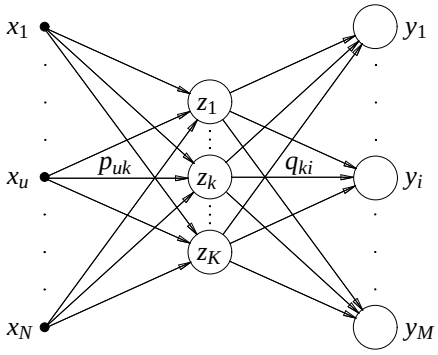

Figure 3: The multilayer perceptron equivalent for the general MF scheme

In the learning phase, an incremental learning method is used. For the (u,i)-th rating, we set the input $\mathbf{x}$ so that x_u is 1 and $x_{u' \neq u} = 0$, thus $z_k = p_{uk}$ holds. Let $\hat{r}_{ui} = y_j$ denote the i-th output of the network. We compute the error: $e_{ui} = r_{ui} - \hat{r}_{ui}$. This error is back-propagated from the i-th output

neuron towards the input layer. This neural network (NN) with this special learning is equivalent to the ISMF.

If we apply regularization (weight decay), we get the RISMF approach.

We can extend this NN to be equivalent to the BRISMF: item biases can be handled by adding a bias (constant 1) input to the output neurons; user biases can be handled by setting $\mathbf{q}_{2\bullet}$ weights to 1 and keeping them constant. In the evaluation phase, we set the input as in the learning phase, and y_i $(i = 1,\ldots,M)$ predicts the active user's rating on the i-th item.

It was shown that feed-forward neural networks with linear activation function can find the principal components of a data set (Baldi and Hornik, 1989), since all local minima are global minima there. However, our problem setting differs in two minor and one major issues: First, our goal is to factorize a non-squared matrix (the number of inputs of the neural network is different from the number of outputs), second, we penalize the weights of the neural network, and principally, not all output are defined, since users rate only a small subset of the items. This latter point infers that we cannot expect local minima to be global minima, as it has been reflected on page 629.

4. Neighbor Based Correction of MF

Neighbor based (NB) approaches exploit the observation that similar users rate similar items similarly. In the NB scheme a set of similar users is selected for each query from among those who rated the active item. Or analogously, a set of similar items is selected from among those that have been rated by the active user. The answer of the predictor is then obtained by combining the ratings of similar users (items) for the active item (user). The first variant is termed the user neighbor based, and the second is termed the item neighbor based approach.

MF and 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 2 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 (Bell and Koren, 2007a; Bell et al., 2007b). However, the price of additional accuracy is paid by the decreased scalability. Here we propose a scalable solution for unifying the MF and NB approaches. We will point out that it is computationally less expensive than Bell et al's approaches.

The idea is that we try to improve an existing MF model (**P**,**Q**) by adding a neighbor based correction term to its answer in the prediction phase. Assuming the item neighbor based scheme, the corrected answer for query (u,i) is the following:

$$\hat{r}_{ui} = \mathbf{p}_u^T \mathbf{q}_i + \gamma \frac{\sum_{j \in \mathcal{T}_u \setminus \{i\}} s_{ij} \left(\mathbf{p}_u^T \mathbf{q}_j - r_{uj} \right)}{\sum_{j \in \mathcal{T}_u \setminus \{i\}} s_{ij}},$$

where s_{ij} is the similarity between $\mathbf{q}_i$ and $\mathbf{q}_j$, and $\mathcal{T}_u$ is set of the items rated by user u. The weight of the correction term γ can be optimized via cross-validation. This model can be seen as a unification of the MF and NB approaches.

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

The similarity s_{ij} can be defined in many different ways. Here we propose two variants.

- (S1): Normalized scalar product based similarity.

$$s_{ij} = \left(\frac{\max\{0, \sum_{k=1}^{K} q_{ki} q_{kj}\}}{\sqrt{\sum_{k=1}^{K} q_{ki}^2} \cdot \sqrt{\sum_{k=1}^{K} q_{kj}^2}} \right)^{\alpha},$$

  where α is an amplification parameter.

- (S2): Normalized Euclidean distance based similarity.

$$s_{ij} = \left(\frac{\sum_{k=1}^{K} (q_{ki} - q_{kj})^2}{\sqrt{\sum_{k=1}^{K} q_{ki}^2} \cdot \sqrt{\sum_{k=1}^{K} q_{kj}^2}} \right)^{-\alpha}.$$

In both cases, the value s_{jk} can be calculated in O(K) time, thus $\hat{r}_{ui}$ can be calculated in O($K \cdot |\mathcal{T}_u|$). We remark that one can restrict to use only the top S neighbors of the queried item (Bell and Koren, 2007a), 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 requirements of our NB corrected MF in comparison with the improved neighborhood based approach of Bell and Koren (2007a), which can also be applied to further improve the results of an MF. For a given query, the running time 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 u-th user, our method requires the storing $\mathbf{q}_u$ and $\mathbf{Q}$ in the memory, that is O(KN), while their approach must store the item-by-item matrix and the ratings of the user, which is O($M^2 + |\mathcal{T}_u|$). For all users, our method requires O($NK + KM$) while their approach requires O($M^2 + |\mathcal{T}|$) memory.

Despite the simplicity of our method its effectiveness is comparable with that of Bell and Koren's method, see Section 5.4.9. We highly recommend this method as it improves accuracy significantly, as we show above and in Section 5.4.8.

This model can be seen as simple, scalable, and accurate unification of the MF and NB approaches. 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.

5. Experiments

Currently, the largest publicly available ratings data set is provided by Netflix, a popular online DVD rental company. Netflix initiated the Netflix Prize contest in order to improve their recommender system—called Cinematch—that provides movie recommendations to customers. The data set released for the competition was substantially larger than former benchmark data sets and contained about 100 million ratings from over 480k users on nearly 18k movies (see details in Subsection 5.1.1). For comparison, the well-known EachMovie data set[4] only consists of 2,811,983 ratings of

4. It used to be available upon request from Compaq, but in 2004 the proprietary retired the data set, and since then it has no longer been available for download.

72,916 users and 1,628 movies. Rashid et al. (2006) used a 3 millions ratings subset of the GroupLens project, which entirely contains about 13 million ratings from 105k users on 9k movies. We evaluated all of our algorithms against the Netflix data set, since currently this is the most challenging problem for the collaborative filtering community, and our work was greatly motivated by it. In addition, to illustrate the applicability of the presented methods on other data sets, we also performed experiments with a selected MF, BRISMF, on the 1M MovieLens and the Jester data set (see details in Subsection 5.1.2–5.1.3).

The evaluation metrics of recommender systems can greatly vary depending on the characteristics of the data set (size, rating density, rating scale), the goal of recommendation, the purpose of evaluation (Herlocker et al., 2004). In the current CF setting the goal is to evaluate the predictive accuracy, namely, how closely the recommender system can predict the true ratings of the users, measured in terms of root mean squared error.

In case of MovieLens and Jester data sets, we also provide the mean absolute error (MAE) since this is the most common performance measure for these sets:

$$\text{MAE} = \frac{1}{|\mathcal{V}|} \sum_{(u,i)\in\mathcal{V}} |\hat{r}_{ui} - r_{ui}|.$$

5.1 Data Sets

In this section we describe in details the above mentioned three data sets.

5.1.1 THE NETFLIX PRIZE DATA SET

The data set provided generously by Netflix for the NP competition contains (u,i,r_{ui},d_{ui}) rating quadruples, representing that user u rated item i as r_{ui} on date d_{ui}, where $d_{ui} \in \mathcal{D}$ the ordered set of possible dates. The ratings r_{ui} are integers from 1 to 5, where 1 is the worst, and 5 is the best. The data were collected between October, 1998 and December, 2005 and reflect the distribution of all ratings received by Netflix during this period (Bennett and Lanning, 2007). The collected data was released in a train-test setting in the following manner (see also Figure 4).

Netflix selected a random subset of users from their entire customer base with at least 20 ratings in the given period. A *Hold-out set* was created from the 9 most recent ratings of the users, consisting of about 4.2 million ratings. The remaining data formed the Training set. The ratings of the Hold-out set were split randomly with equal probability into three subsets of equal size: Quiz, Test and Probe. The *Probe set* was added to the Training set and was released with ratings. The ratings of the *Quiz and Test sets* were withheld as a *Qualifying set* to evaluate competitors. The Quiz/Test split of the Qualifying set is unknown to the public. We remark that the date based partition of the entire NP data set into train-test sets reflects the original aim of recommender systems, which is the prediction of future interest of users from their past ratings/activities.

As the aim of the competition is to improve the prediction accuracy of user ratings, Netflix adopted RMSE as evaluation measure. The goal of the competition is to reduce by at least 10 percent the RMSE on the Test set, relative to the RMSE achieved by Cinematch.[5] The contestants have to submit predictions for the Qualifying set. The organizers return the RMSE of the submissions on

5. The first team achieving the 10 percent improvement is promised to be awarded by a grand prize of $1 million by Netflix. Not surprisingly, this prospective award drawn much interest towards the competition. So far, more than 3 000 teams submitted entries for the competition.

the Quiz set, which is also reported on a public leaderboard.[6] Note that the RMSE on the Test set is withheld by Netflix.

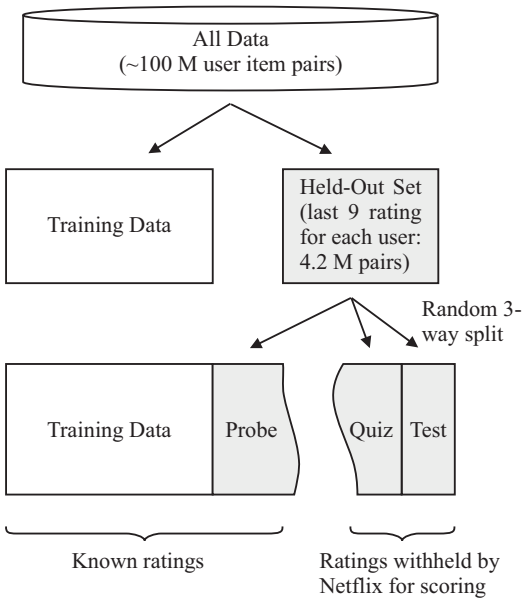

Figure 4: The train-test split and the naming convention of Netflix Prize data set, after Bell et al. (2007a)

There are some interesting characteristics of the data and the set-up of the competition that pose a difficult challenge for prediction:

- The distribution over the time of the ratings of the Hold-out set is quite different from the Training set. As a consequence of the selection method, the Hold-out set does not reflect the skewness of the movie-per-user, observed in the much larger Training set. Therefore the Qualifying set contains approximately equal number of queries for often and rarely rating users.

- The designated aim of the release of the Probe set is to facilitate unbiased estimation of RMSE for the Quiz/Test sets despite of the different distributions of the Training and the Hold-out sets. In addition, it permits off-line comparison of predictors before submission.

- We already mentioned that users' activity at rating is skewed. To put this into numbers, ten percent of users rated 16 or fewer movies and one quarter rated 36 or fewer. The median is 93. Some very active users rated more than 10,000 movies. A similar biased property can be observed for movies: The most-rated movie, *Miss Congeniality* was rated by almost every second user, but a quarter of titles were rated fewer than 190 times, and a handful were rated fewer than 10 times (Bell et al., 2007a).

6. Found at `http://www.netflixprize.com/leaderboard`.

- The variance of movie ratings is also very different. Some movies are rated approximately equally by the user base (typically well), and some partition the users. The latter ones may be more informative in predicting the taste of individual users.

In this experimentation section we evaluate the presented methods on a randomly selected 10% subset of the Probe set, which we term as Probe10.[7] Unless we explicitly mention, from now on the RMSE values refer to the Probe10 RMSE. We have decided to report on RMSE values measured on the Probe10 set, since in our experiments the Probe10 RMSE are significantly closer to the Quiz RMSE than Probe RMSE, and Quiz RMSE tell us more about the accuracy of the predictor, since it excludes the impact of overtraining. On the other hand the rules of NP competition allows only 1 submission daily, which limits the number of the Quiz RMSE calculation drastically. We remark that we measured typically 0.0003 difference between the Probe10 and the Quiz RMSE values (sometimes 0.0010), while this was of an order of magnitude larger for the Probe set. This advantageous property nominates the Probe10 set for being a standard and Netflix-independent evaluation set for predictors trained on the NP data set.

We performed a thorough analysis to check how reliable the Probe10 results are. For this, we will show that if a given method has better RMSE compared to another method for a particular subset of the Probe set, then it has the same performance gain on other subsets of the Probe set. To do this, we partitioned the Probe set into 10 subsets, and we ran 10 different methods using them. Therefore, in total we had 100 runs. We denote the test RMSE of the x-th method on the y-th test set by m_{xy}. To summarize the results, we calculated the average test RMSE for each method, denoted by $(\overline{m}_{1\bullet},\ldots,\overline{m}_{10\bullet})$, and a *difficulty offset* for each test set, denoted by $(o_1,\ldots,o_{10})$, defined as

$$o_y = \frac{1}{10}\sum_{x=1}^{10} m_{xy} - \overline{m}_{x\bullet}.$$

The test RMSE of the 100 runs are approximated as $m_{xy} = \overline{m}_{x\bullet} + o_y$. The standard deviation of this approximation is 0.000224 RMSE score, and the maximal deviation is less than 0.0007 RMSE score, which means that m_{xy} is well approximated, thus we can assign a difficulty offset to each test set. Consequently, our initial hypothesis is verified.

When Quiz RMSE values are reported we also mention the percentage of improvement over Cinematch (IoC). We performed all tests on an average single processor laptop (a 2 GHz Intel Pentium M (Dothan) with 1 GB RAM), on which reported training times were measured.

5.1.2 MOVIELENS DATA SET

The 1M MovieLens data set[8] contains cca. 1 million ratings from 6,040 users on 3,900 movies. As in the case of NP, ratings are made on a 5 star scale, and the rating records are also quadruples containing the timestamp of the rating. Demographic data provided with the ratings are not used in our setting. Since there is no standard train-test split of the data set, we applied a simple random split to generate a 90%–10% train-test setting.[9]

7. A Perl script can be downloaded from our homepage, `http://gravityrd.com`, which selects the Probe10 from the original Netflix Probe set to ensure repeatability and comparability.
8. Available at: `http://www.grouplens.org/node/73`.
9. This split can be downloaded from our website: `http://gravityrd.com`.

5.1.3 JESTER DATA SET

The Jester data set[10] (Goldberg et al., 2001) contains 4,136,360 ratings from 73,421 users on 100 jokes. Users rated jokes on the continuous $[-10, +10]$ range. Ten percent of the jokes (called the gauge set, which users were asked to rate) are densely rated, others, more sparsely. Two thirds of the users have rated at least 36 jokes, and the remaining ones have rated between 15 and 35 jokes. The average number of ratings per user is 46, so it is a particularly dense data set compared to NP and MovieLens. Goldberg et al. (2001) created their train-test split by a random division of a subset of 18,000 users into two disjoint sets. For our experiments this split is obviously inappropriate since it does not enable us to integrate user preferences into the model. Therefore, here we also applied a random split to generate a 90%–10% train-test setting.[11]

5.2 Implementation Issues

Because the data set is huge, its storage is an important issue. This was thoroughly investigated in our previous work (see Takács et al., 2007). We have shown there that the entire data set can be stored in 300 MB without storing the dates but keeping the chronological order of the ratings, and in 200 MB without even keeping the order. This enables to perform the algorithms on an average PC (our platform details are given at the end of Section 5.1.1). As we will point out here, MF methods are sensitive to the order of training examples and the selection of their proper order exploits date information. On the other hand, pure NB approaches are not sensitive to the order of training examples.

The users' tastes change in time, and when providing them recommendations, only their current taste matters. This phenomenon is modeled in the NP data set as the value of d_{ui} is greater for test examples than for training examples. We can condition MF methods to exploit the date information by properly ordering training examples. We found the following order to be very effective: iterate over users in an arbitrary order, and for each user, take the ratings in an increasing chronological order, that is, starting from the oldest and ending with the newest. Unless explicitly mentioned otherwise, in our experiments we use this training example order for MF methods. The impact of the order on the accuracy of MFs is investigated in Subsection 5.4.4.

5.3 Parameter Optimization

All of the presented methods have many pre-defined parameters that greatly influence the accuracy. Sometimes a few experiments are enough to set them well, sometimes we need to apply parameter optimization to find the best settings.

We recall that a parameter setting can be advantageous because (1) it produces a very accurate model (low validation RMSE) or (2) it "blends well", that is it improves the accuracy of the blended model. The more parameters a method has, the harder to set them well, but the more chance to get a better RMSE.

We used random search and Algorithm 3 to optimize parameters. The typical value of n is 2. In case of MF, we have experimented with many parameters, namely:

- the number of features: K;
- different learning rate and regularization factor

10. Available at: `http://goldberg.berkeley.edu/jester-data/`.
11. This split can be downloaded from our website: `http://gravityrd.com`.

Input: L, $p_1, \ldots, p_L$, n
Output: $v_1, \ldots, v_l$
Randomly initialize parameters $p_1, \ldots, p_L$
Iterate forever (iteration is stopped manually).
  Randomly choose one parameter: p_l;
  Randomly generate n different values
    to that parameter: $u_1, \ldots, u_n$;
  Let u_0 be the current value of the p_l.
  For each of the $u_0, \ldots, u_n$ values, run a training
    algorithm, temporarily setting p_l to that value, and
    evaluating the model on the validation set.
  Assign the best value to p_l.
end

Algorithm 3: Simple parameter optimization algorithm

  - for users and movies ($\eta^{(p)}, \eta^{(q)}, \lambda^{(p)}, \lambda^{(q)}$);
  - for the corresponding variables of bias features ($\eta^{(pb)}, \eta^{(qb)}, \lambda^{(pb)}, \lambda^{(qb)}$);
- minimum and maximum weights in the uniform random initialization of **P** and **Q**: $w_{\underline{p}}, w_{\overline{p}}, w_{\underline{q}}, w_{\overline{q}}$;
- G: the offset to subtract from **R** before learning (can be, for example, set to the global average of ratings);

We subsampled the matrix **R** for faster evaluation of parameter settings. We have experienced that movie-subsampling substantially increased the error, in contrast to user-subsampling, thus we do not perform the former. The reason for this is that in the evaluation data set movies have much more ratings than users. Consequently, if we do user-subsampling for example with 100 instead of the original 200 ratings we lose more information than at movie-subsampling when we have for example 10000 ratings instead of 20000. Interestingly, the larger the subsample is, the fewer iterations are required to achieve the optimal model. This can be explained by the existing redundancy in the data set. This implies also that the time-complexity of MF is sublinear in the number of ratings.

5.4 Results of Matrix Factorization

We recall that we applied linear combinations of methods for blending,[12] and we ordered the training examples user-wise and then by date, as specified in Section 5.2.

5.4.1 COMPARING REGULARIZED AND BIASED MF VARIANTS

We compare:

- an instance of the regularized RISMF, termed as RISMF#0, with the following parameter settings: $K = 40, \eta = 0.01, \lambda = 0.01, w_{\underline{p}} = -w_{\overline{p}} = w_{\underline{q}} = -w_{\overline{q}} = -0.01$
- an instance of the biased BRISMF, named briefly as BRISMF#0, with the following parameter settings: $K = 40, \eta = 0.01, \lambda = 0.01, w_{\underline{p}} = -w_{\overline{p}} = w_{\underline{q}} = -w_{\overline{q}} = -0.01$

12. The source code of our combination algorithm can be downloaded from our web site: `http://gravityrd.com`.

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

5.4.2 CHANGING THE PARAMETERS OF THE BRISMF

Table 1 presents the influence of η and λ on the Probe10 RMSE and the optimal number of epochs. Other parameters are the same as in BRISMF#0. The best result is RMSE $= 0.9056$ when $\eta = 0.007, \lambda = 0.005$, which is a 0.0057 improvement. We refer to this MF in the following as BRISMF#1. The running time for this MF is only 14 minutes! Note that running time depends only on K, and on the optimal number of epochs.

Note that decreasing η or increasing λ increases the optimal number of epochs, (except for $\eta = \lambda = 0.020$).

η \ λ	0.005	0.007	0.010	0.015	0.020
0.005	0.9061 / 13	0.9079 / 15	0.9117 / 19	0.9168 / 28	0.9168 / 44
0.007	**0.9056 / 10**	0.9074 / 11	0.9112 / 13	0.9168 / 19	0.9169 / 31
0.010	0.9064 / 7	0.9077 / 8	0.9113 / 10	0.9174 / 13	0.9186 / 21
0.015	0.9099 / 5	0.9111 / 6	0.9152 / 6	0.9257 / 7	0.9390 / 7
0.020	0.9166 / 4	0.9175 / 4	0.9217 / 4	0.9314 / 4	0.9431 / 3

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

Now we show that the usage and proper setting of new parameters can boost the performance: we introduce the parameters $\eta^{(p)}, \eta^{(q)}, \lambda^{(p)}, \lambda^{(q)}$ and $\eta^{(pb)}, \eta^{(qb)}, \lambda^{(pb)}, \lambda^{(qb)}$ (see Section 5.3 for explanation).

Initially $\eta^{(p)} = \eta^{(q)} = \eta^{(qb)} = \lambda^{(pb)} = 0.007$, and $\lambda^{(p)} = \lambda^{(q)} = \lambda^{(pb)} = \lambda^{(qb)} = 0.005$, to yield the RMSE $= 0.9056$ given above. Finding the best setting of these 8 variables is practically impossible and can cause overlearning on Probe10. To demonstrate how the parameter optimization algorithm mentioned in Section 5.3 works, we apply its simplified version: we do not use random numbers, just fix the order of these variables and define the possible values for them. Let the order be: $\eta^{(p)}, \eta^{(q)}, \eta^{(pb)}, \lambda^{(qb)}, \lambda^{(p)}, \lambda^{(q)}, \lambda^{(pb)}, \lambda^{(qb)}$. Let the set of values for η variants be $\{0.005, 0.007, 0.010\}$, and for the λ variants: $\{0.003, 0.005, 0.007\}$. Table 2 shows step-by-step how parameters are optimized one-by-one in 8 iterations. The parameter optimization procedure decreased the RMSE score from 0.9056 to 0.9036.

5.4.3 BRISMF RESULTS ON MOVIELENS AND JESTER DATA SETS

We performed several tests on MovieLens and Jester data sets with the BRISMF method. As mentioned earlier there are no standard train-test split for these data sets; therefore it is difficult to compare our obtained results with already published ones. Consequently, we used three baseline methods for comparison. The *constant* method always predicts the average of the ratings in the training set, the *item average* outputs the average of the training ratings of the active item at querying, while the *item neighbor* (Takács et al., 2008b) is an item neighbor based method with Pearson

	0.003	0.005	0.007	0.010	Decision
$\eta^{(p)}$		0.9057	**0.9056**	0.9058	$\eta^{(p)} := 0.007$ (no change)
$\eta^{(q)}$		0.9057	**0.9056**	0.9061	$\eta^{(q)} := 0.007$ (no change)
$\eta^{(pb)}$		0.9059	0.9056	**0.9052**	$\eta^{(pb)} := 0.010$
$\eta^{(qb)}$		0.9053	**0.9052**	0.9056	$\eta^{(qb)} := 0.007$ (no change)
$\lambda^{(p)}$	0.9053	0.9052	**0.9051**		$\lambda^{(p)} := 0.007$
$\lambda^{(q)}$	0.9057	0.9051	**0.9050**		$\lambda^{(q)} := 0.007$
$\lambda^{(pb)}$	0.9053	0.9050	**0.9047**		$\lambda^{(pb)} := 0.007$
$\lambda^{(qb)}$	**0.9036**	0.9047	0.9066		$\lambda^{(qb)} := 0.003$

Table 2: Effect of parameter optimization on BRISMF#1

correlation based similarity. The MAE results for BRISMFs were obtained by using the same **P** and **Q** that were used to get the RMSE values.

For MovieLens, the main training parameters were set to $\eta^{(p)} = \eta^{(q)} = 0.01$, $\lambda^{(p)} = \lambda^{(q)} = 0.02$, and the number of features (K) was varied as tabulated in Table 3. The obtained results show that the increase of the number of features K yields better accuracy at a decreasing number of epochs.

Model	Epochs	RMSE w/o S2	RMSE with S2	MAE w/o S2	MAE with S2
constant	–	1.1179	–	0.9348	–
item average	–	0.9793	–	0.7829	–
item neighbor	–	0.8521	–	0.6641	–
BRISMF#5	35	0.8555	0.8537	0.6684	0.6667
BRISMF#10	27	0.8471	0.8426	0.6608	0.6563
BRISMF#20	24	0.8435	0.8363	0.6582	0.6507
BRISMF#50	23	0.8396	0.8319	0.6544	0.6461
BRISMF#100	24	0.8378	0.8299	0.6531	0.6444
BRISMF#200	21	0.8365	0.8285	0.6519	0.6430
BRISMF#500	20	0.8353	0.8275	0.6508	0.6424

Table 3: Test RMSE of various methods on the MovieLens data set

In terms of RMSE, the simplest BRISMF#5 achieves 12.64% improvement over the item average, while this is 14.70% for the largest BRISMF#500.[13] We also included in the table the test RMSE value achieved with neighbor correction using similarity function S2 ($\alpha = 5$). The improvement over the item average is 12.82% and 15.5% for the neighbor corrected BRISMF#5 and BRISMF#500, respectively. The S2 correction improves RMSE value from 0.0018 to 0.0080, which can be as large as almost 1% improvement over the RMSE of the non-corrected version. One can observe that the percentage of improvement increases with the number of features. The improvement of BRISMF#500 with S2 correction over item neighbor is 2.89%, which is similar to the results published in Takács et al. (2008b) and Takács et al. (2008a) for the Netflix data set.

13. We recall that Cinematch produces 9.6% improvement over item average (Quiz RMSE= 1.0540) on the NP data set (Bennett and Lanning, 2007).

In terms of MAE, the tendency of the improvements is almost identical. The simplest and the largest BRISMF achieves 14.63% and 16.87% improvements, which become 14.84% and 17.95% with NB correction. Here the magnitude of improvement is somewhat larger.

The reported results on RMSE and MAE value on the MovieLens data set are not directly comparable with ours because of the use of different train-test splits. The best known results are given by Delannay and Verleysen (2007): RMSE 0.875 and MAE 0.648 that are obtained with an interlaced generalized linear model. Similar MAE = 0.652 was reported by DeCoste (2006) achieved with ensembles of maximum margin matrix factorizations.

For Jester, the main training parameters were set to $\eta^{(p)} = \eta^{(q)} = 0.002$, $\lambda^{(p)} = \lambda^{(q)} = 0.02$, and K was varied as tabulated in Table 4. The obtained results show that the increase of K yields better accuracy, while the number of epochs is almost the same. Due the different characteristics of the Jester data set, the magnitude of RMSE scores are larger. It is interesting that on the Jester data set the item neighbor method gives better results than BRISMF. We think that this phenomenon is due to the different characteristics of Jester data set when compared to NP and MovieLens data sets: the rating matrix is almost dense and there are only 100 items.

Model	Epochs	RMSE w/o S2	RMSE with S2	MAE w/o S2	MAE with S2
constant	–	5.2976	–	4.4372	–
item average	–	5.0527	–	4.1827	–
item neighbor	–	4.1123	–	3.1616	–
BRISMF#5	7	4.2080	4.1902	3.2608	3.2352
BRISMF#10	8	4.1707	4.1575	3.2201	3.1967
BRISMF#20	7	4.1565	4.1405	3.2095	3.1820
BRISMF#50	8	4.1399	4.1265	3.1876	3.1616
BRISMF#100	7	4.1395	4.1229	3.1909	3.1606

Table 4: RMSE of various methods on the Jester data set

In terms of RMSE, the simplest BRISMF#5 achieves 16.72% improvement over the item average, while this is 18.07% for the largest BRISMF#100. We also included in the table the test RMSE value achieved with neighbor correction using similarity function S2 ($\alpha = 5$). The improvement over the item average is 17.07% and 18.40% for the neighbor corrected BRISMF#5 and BRISMF#100, respectively. The S2 correction improves the RMSE value from 0.0132 to 0.0178, which can be an over 0.4% improvement over the RMSE of the non-corrected version. In terms of MAE, the improvements are somewhat larger; they reach 23.72% without and 24.43% with S2 correction.

Here we also indicate some of the best published RMSE and MAE scores, keeping in mind that those are not directly comparable due to different test settings. Delannay and Verleysen (2007) achieved RMSE 4.17 and MAE 3.26 with an interlaced generalized linear model; the same MAE score was obtained by Canny (2002) with a sparse factor analysis model.

We can conclude from the experiments performed on the MovieLens and Jester data sets that the applicability of MF based methods and neighbor based correction technique is not restricted to the NP data set. Rather, they are useful CF techniques for different problems as well.

5.4.4 ORDER OF EXAMPLES

We examined how the order of examples influences the result by comparing the RMSE of BRISMF#1 on two different orders: For the proposed order the RMSE is 0.9056, and for a random order—obtained by a random shuffle of the ratings of each user—the RMSE is 0.9104.

5.4.5 SUBSAMPLING USERS

On Figure 5 we demonstrate how the number of users (thus, the number of ratings) influences RMSE and the optimal number of training epochs in case of BRISMF#1. 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 RMSE and the number of epochs. This means that the time-complexity of MF is sublinear in the number of ratings (see Section 5.3). We remark that the number of ratings is proportional to the number of users; the ratio of them—which is equal to the average number of ratings per user—is 209 in the training set.

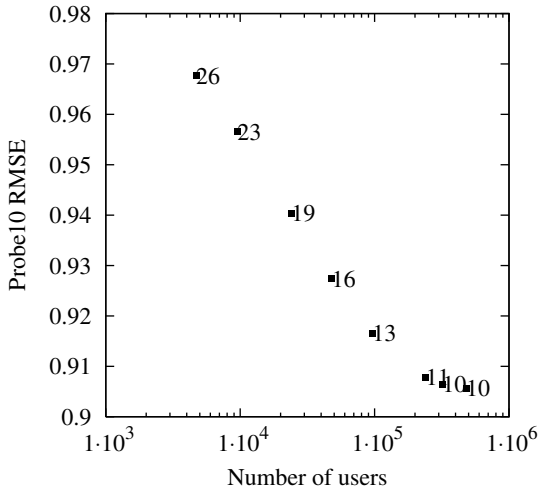

Figure 5: Effect of the number of users on Probe10 RMSE and on the optimal number of training epochs.

5.4.6 SEMIPOSITIVE AND POSITIVE MF

We investigated the accuracy of semipositive and positive variants using the following MFs:

- SemPosMF#800: this is a semipositive MF, where user features are nonnegative and item features are arbitrary. Parameters are set to: $K = 800, w_{\underline{p}} = 0, w_{\overline{p}} = -w_{\underline{q}} = w_{\overline{q}} = 0.005, \eta^{(p)} = \eta^{(pb)} = 0.016, \eta^{(q)} = \eta^{(qb)} = 0.005, \lambda^{(p)} = \lambda^{(q)} = 0.010, \lambda^{(pb)} = \lambda^{(qb)} = 0, G = 3.6043$. After 12 epochs, learning rates are multiplied by 0.01, and the model is trained for another 2 epochs.
- PosMF#400: this is a positive MF. Parameters are the same as in SemPosMF#800, except that $K = 400, w_{\underline{q}} = 0$.
- PosMF#100: like PosMF#400, but $K = 100$.

The results are summarized on Table 5.

Model	Epochs	RMSE
SemPosMF#800	12+2	0.8950
PosMF#400	14	0.9036
PosMF#100	14	0.9078

Table 5: Probe10 RMSE of positive and semipositive MFs

5.4.7 RETRAINING USER FEATURES

We investigated the effectiveness of retraining user features. We tested both solutions proposed in Section 3.5. The experiments were run with three parameter settings: BRISMF#1, and the following two MFs:

- BRISMF#250: $K = 250$, $w_{\underline{p}} = -0.01$, $w_{\overline{p}} = -0.006$, $w_{\underline{q}} = -0.010$, $w_{\overline{q}} = 0.020$, $\eta^{(p)} = 0.008$, $\eta^{(pb)} = 0.016$, $\eta^{(q)} = 0.015$, $\eta^{(qb)} = 0.007$, $\lambda^{(p)} = 0.048$, $\lambda^{(q)} = 0.008$, $\lambda^{(pb)} = 0.019$, $\lambda^{(qb)} = 0, G = 0$.
- BRISMF#1000: the same as BRISMF#250, but $K = 1000$.

First, we investigated the first solution (intra-training user-wise test). For BRISMF#250 the obtained Probe10 RMSE is 0.8959, which is only a slight 0.0002 improvement.

Second, we tested the second solution (see Table 6). Both the simpler case (after the first learning step, reset **P** and retrain only **P**), and the advanced case (after the first learning step, reset **P** and retrain both **P** and **Q**) are analyzed. We append letter "U" to the method name in the simpler case (BRISMF#1 becomes BRISMF#1U, etc.) and letters "UM" in the advanced case (BRISMF#1UM, etc.). We indicated the required number of epochs 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 improves their performance. BRISMF#1000UM is currently our best MF: Probe10 RMSE is 0.8921, Quiz RMSE is 0.8918.

Model	Epochs	Probe10	Quiz	IoC
BRISMF#1	10	0.9056		
BRISMF#1U	10+8	0.9072		
BRISMF#1UM	10+6	0.9053		
BRISMF#250	14	0.8961	0.8962	5.80%
BRISMF#250U	14+8	0.8953	0.8954	5.89%
BRISMF#250UM	14+7	0.8937		
BRISMF#1000	14	0.8938	0.8939	6.04%
BRISMF#1000U	14+8	0.8936		
BRISMF#1000UM	14+8	0.8921	0.8918	6.26%

Table 6: Examining the effect of retraining user features

We investigated the effect of retraining only **P**, when BRISMF#250 was learnt on a subset of the database. The question in this case is: how reliable is a **Q** learnt only on a subset of the database.

First, we kept only the 40%, 60% or 80% of users and ran an MF algorithm and fixed the resulting **Q**. Then we reset and learnt **P**, first on the same subset of the database, and then on the

entire database. In all 3 cases, the difference between the two Probe10 RMSE results was less than 0.0013. Each Probe10 RMSE was less than 0.8970. Thus, we can conclude that the proposed retraining method can handle the addition of new users.

Second, we discarded the last N_1 ratings of each user and ran the same retraining procedure. In our experiments, N_1 was set to 0, 10, 20, 40. Obviously, the removal of ratings increased Probe10 RMSE significantly; the highest score was 1.0038, whereas Probe10 RMSE on the entire training database went up to only 0.8980, which means that the **Q** calculated on the subset of the data differs slightly from the **Q** calculated on the entire data set. Thus, the proposed retraining method can handle the addition of new ratings as well. These experiments verify the usability of user feature retraining method for handling new users or ratings.

5.4.8 THE EFFECT OF CORRECTION TECHNIQUES

In order to investigate the effect of various correction techniques, we first generated a number of accurate MF models. We will show that correction techniques improve accuracy significantly for all of these models. We applied the parameter optimization method mentioned in section 5.3 to get accurate MFs. Also, we applied the "trial and error" method to get manually parameterized accurate MFs. Here we describe some results of both:

- BRISMF#800: manually parameterized MF, with 800 features. Parameters are set to: $K = 800$, $w_{\underline{p}} = -w_{\overline{p}} = w_{\underline{q}} = -w_{\overline{q}} = -0.005, \eta^{(p)} = \eta^{(pb)} = 0.016, \eta^{(q)} = \eta^{(qb)} = 0.005, \lambda^{(p)} = \lambda^{(q)} = 0.010, \lambda^{(pb)} = \lambda^{(qb)} = 0, G = 3.6043$. After 9 epochs, learning rates are multiplied by 0.01, and the model is trained for another 2 epochs.

- SemPosMF#800: defined in Section 5.4.6.

- MlMF#200: a BRISMF with 200 features. Parameters are found by the parameter optimization algorithm.

- MlMF#80: a BRISMF with 80 features. Parameters are found by the parameter optimization algorithm.

- MomentumMF: a BRISMF with momentum method, manually optimized: $K = 50$, $\eta = 0.01$, $\sigma = 0.3$ and $\lambda = 0.00005$. Model learnt in 5 epochs.

We refer to a variant of the transductive MF algorithm as Q-correction: in Eq. (9) to improve predictions we use only the ratings in the Qualify set, not in the Probe10 + Qualify set. See Table 7 for the RMSE values of the each method and its blended versions. We applied two NB corrections to the MF models, with similarities S1 and S2.

The results indicated in the Q, S1, S2 and Q + S1 + S2 columns are obtained by using one or more correction techniques; thus those figures refer to linear regression of predictors (columns) on Probe10 data. Each correction technique adds one more column to the combination of the basic method; that is Q, S1, S2 add 1 extra column, Q + S1 + S2 adds 3 extra columns.

One can observe in Table 7 that NB correction significantly improves the result of MF based methods. Starting from an average MF (MlMF#80) the reduction of RMSE can be 0.0179, it reduces the RMSE of the good MomentumMF by 0.0075, and it even improves slightly (0.0026) the very accurate BRISMF#800. We recall that we measured similar accuracy improvement using NB correction (with S2 similarity) in the case of the MovieLens and the Jester data sets (see Sections 5.1.2

#	Model	basic	Q	S1	S2	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
3	MlMF#200	0.9112	0.9106	$0.9087_{(\alpha=8)}$	$0.9085_{(\alpha=6)}$	0.9076
4	MlMF#80	0.9251	0.9240	$0.9104_{(\alpha=9)}$	$0.9072_{(\alpha=2)}$	0.9058
5	BRISMF#1000UM	0.8921	0.8918	$0.8905_{(\alpha=7)}$	$0.8907_{(\alpha=5)}$	0.8901
6	MomentumMF	0.9031	0.9020	$0.8979_{(\alpha=6)}$	$0.8956_{(\alpha=3)}$	0.8949
7	1 + 2	0.8923				0.8880
8	1 + 2 + 3	0.8913				0.8872
9	1 + 2 + 3 + 4	0.8909				0.8863
10	1 + 2 + 3 + 4 + 5	0.8895				0.8851
11	1 + 2 + 3 + 4 + 5 + 6	0.8889				0.8838

Table 7: Probe10 RMSE of accurate MFs without and with applying Q-correction and NB correction (S1 and S2). At columns S1 and S2 we also indicated the optimal value of parameter α.

and 5.1.3). In comparison, BellKor's approach (Bell and Koren, 2007a, 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 correction techniques, which is a linear combination of 24 methods, then the combination yields RMSE = 0.8838, Quiz RMSE = 0.8839. Using only the first 4 methods with all corrections (combination of 16 methods), it yields RMSE = 0.8863, Quiz RMSE = 0.8862.

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 BRISMF#800, then the RMSE increases only by 0.0011 to 0.8850. In general, we can state that one "correction technique" brings a 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-correction and NB corrections captures the same aspects of the data, regardless of the MF behind them.

These experiments demonstrate that the Probe10 set containing 140,840 ratings is big enough to evalute not only single methods, but also combinations of many methods.

5.4.9 COMPARISON WITH BELLKOR'S POSTPROCESSING

The neighbor based correction of MF can also be done by running a neighbor based method on the residuals of MF. A very effective known algorithm for postprocessing the residuals of MF is BellKor's neighbor based method (Bell and Koren, 2007b) (BKNB). The comparison of our neighbor correction scheme and BKNB can be seen in Table 8.

In the experiments we applied the techniques on the residuals of 3 models: a BRISMF#100, a SemPosMF#100, and a so called *global effects model* (Bell and Koren, 2007b) with 12 effects. For running S1 and S2 correction on global effects we used the item features of SemPosMF#100.

Model	basic	S1	S2	S1 + S2	BKNB
BRISMF#100	0.8979	0.8940	0.8937	0.8933	0.8948
SemPosMF#100	0.9001	0.8954	0.8951	0.8946	0.8957
GlobalEffects#12	0.9600	0.9196	0.9237	0.9174	0.9145

Table 8: Comparison of NB correction and BKNB in terms of Probe10 RMSE.

For MF models the most accurate postprocessing technique is S2 correction. In the case of global effects BKNB gives the lowest Probe10 RMSE. It is also important to mention that our approach is significantly faster than BKNB (see Section 5.4.10).

5.4.10 SPEED VS. ACCURACY

From the scalability point of view, it is interesting and important to investigate the relationship of speed and accuracy. We ran numerous randomly parameterized MFs with $K = 40$, and collected the best accuracies in each epoch, and then optimized the parameters. Table 9 summarizes the results. One epoch takes 80 seconds ($K = 40$), and the initialization takes an additional 40 seconds (loading the full database into the memory).

An 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)! For a comparison: Netflix's Cinematch algorithm can achieve Quiz RMSE 0.9514, so this fast solution achieves more than 5.6% improvement on Cinematch.

In Table 9, 1.1 epoch means that the model was trained for one epoch and then the ratings of the first 1/10 of users was used for another epoch. The reason is the same as for feature retraining (see Section 3.5): when we train only for 1 epoch, the features of the first trained users will be obsolete at the end of the epoch, since items have nonsense values at the beginning of the training procedure, and item features change significantly by the end of the epoch.

Epoch	Training Time (sec)	RMSE
1	120	0.9179
1.1	128	0.9147
2	200	0.9071
3	280	0.9057
4	360	0.9028
5	440	0.9008
6	520	0.9002

Table 9: Probe10 RMSE and running time of fast and accurate MFs.

To our best knowledge, the running times and accuracies here and in the previous sections are favorable compared to any other method published in the field of Collaborative Filtering. Though this statement might seem somewhat speculative since authors do not tend to publish running times, we can support it with the following arguments:

- we train each feature simultaneously;
- the number of epochs is small;

- we use a gradient descent algorithm, which is the fastest if we can keep the number of required gradient steps low, which is exactly the case.

Note that given n^*, the number of epochs, there are $n^* \cdot |\mathcal{T}| \cdot K$ variable updates in **P** and **Q** during the training. The presented methods can achieve the favorable RMSEs while keeping the number of features (K) and the number of epochs (n^*) low; consequently they are also favorable in terms of time requirement.

Let us compare the time requirement of our MF methods (all major variants) to one of the best published ones. Bell and Koren (2007a) provide a detailed description of their alternating least squares approach proposed to matrix factorization. Briefly, their idea is to initialize **P** and **Q** randomly, recompute one of them using a nonnegative or a regular least squares solver while the other is constant, then recompute the other, and iterate these two alternating steps for a certain number of epochs (n^*). In the case of the **P**-step, one needs to run the solver for each user to determine how the features of the items rated by the user should be combined to best predict the ratings. One run of the solver requires $\Omega(K^3)$ time, which should be run for each user; thus the **P**-step requires $\Omega(|N| \cdot K^3)$ time.[14] Analogously, the **Q**-step requires $\Omega(|M| \cdot K^3)$ time. The K^2 elements of the covariance matrix need to be updated for each rating, thus, in both alternating steps we update K^2 elements $|\mathcal{T}|$ times. Let n^* denote the optimal number of epochs, which is a few dozen according to their paper. In total, their method requires $\Omega((|N|+|M|) \cdot K^3 + |\mathcal{T}| \cdot K^2) \cdot n^*$ time.

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

Here we neglected the cost of parameter optimization. Our MF has 13 parameters. We perform the parameter optimization process (Sec. 5.3) 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 $\sim$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 it was to tune the parameters to get the same Probe10 RMSE. Consequently, we introduced some additional parameters, for example $\eta^{(p)}, \eta^{(q)}, \eta^{(pb)}, \eta^{(qb)}$ instead of a single η.

5.5 RMSE Values Reported by Other Authors

Finally, let us compare the accuracy of our method (in terms of Probe10 RMSE values that differ from Quiz RMSE values at most by 0.0003) with other RMSE values reported for the Netflix Prize data set. This comparison is difficult since authors often report on RMSE values measured on various custom test sets, different from the Probe and Quiz set. Of the latter two options, Probe RMSE values, which are calculated by leaving out the Probe set from the Train set, can be also misleading, and, consequently, Probe RMSE is often much lower than Quiz RMSE. We remark that Quiz RMSE is often computed by incorporating the Probe data into the training of the predictor. The comparison presented in Table 10 therefore focuses on methods where Quiz RMSE values are available. The table shows that our presented MF method and correction techniques compare favorably with other published ones.

14. There exists somewhat better least squares solvers, but this does not significantly change this comparison.

Source	Method's name	Quiz	IoC	Probe10
Paterek (2007)	Basic + RSVD + RSVD2	0.9070	4.67%	
Salakhutdinov and Mnih (2008)	PMF + PMF with a learnable prior + constrained PMF	0.8970	5.72%	
Bell et al. (2007b)	best stand-alone positive MF	0.9039	4.99%	
	best NB corrected positive MF	0.8953	5.90%	
this paper	stand-alone MF, BRISMF#1000	0.8939	6.04%	0.8938
	stand-alone MF with retrained features, BRISMF#1000UM	0.8918	6.26%	0.8921
	NB corrected MF, BRISMF#1000UM + S1	0.8904	6.41%	0.8905
	stand-alone positive MF, PosMF#400	0.9046	4.92%	0.9036

Table 10: Comparison of Quiz RMSE values of reported MF based methods. We also indicate the Probe10 values of our methods

6. Conclusions

This paper surveyed our approaches for collaborative filtering. We presented several MF methods and a neighbor based correction to the MF. Our methods apply a number of small modifications compared to already published MF variants, but these modifications are together important from the aspects of implementation (time and memory requirements) and accuracy. We performed a comprehensive evaluation of our methods on the Netflix Prize data set, and we showed that the methods can be efficiently applied for other data set (we tested on MovieLens and Jester data sets). We also presented different "correction techniques" to improve prediction accuracy: Q-correction use information from unlabeled examples, while neighbor based correction exploits localized information at prediction. We showed that linear combination of various methods can significantly improve the accuracy of the blended solution. We pointed out that various correction techniques can bring major improvement in accuracy when applied to only one method of the linear combination. We showed that they compare favorably with existing methods in terms of prediction accuracy measured by RMSE and time complexity. The experiments prove that the proposed methods are scalable to large recommender systems having hundreds of millions of ratings.

Acknowledgments

We would like to thank the unknown reviewers for their valuable suggestions and critique. The authors are also grateful for the final thorough proofreading to Lester Mackey. Domonkos Tikk was partly supported by the János Bolyai Research Scholarship of the Hungarian Academy of Science.

References

G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17:734–749, 2005.

P. Baldi and K. Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2(1):53–58, 1989.

R. Bell, Y. Koren, and Ch. Volinsky. Chasing $1,000,000: How we won the Netflix Progress Prize. *ASA Statistical and Computing Graphics Newsletter*, 18(2):4–12, December 2007a.

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

R. M. Bell and Y. Koren. Improved neighborhood-based collaborative filtering. In *Proc. of KDD Cup Workshop at SIGKDD-07, 13th ACM Int. Conf. on Knowledge Discovery and Data Mining*, pages 7–14, San Jose, California, USA, 2007b.

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

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, California, USA, 2007.

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

J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proc. of UAI-98, 14th Conf. on Uncertainty in Artificial Intelligence*, pages 43–52, Madison, Wisconsin, USA, 1998.

J. Canny. Collaborative filtering with privacy via factor analysis. In *Proc. of SIGIR-02, 25th ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 238–245, Tampere, Finland, 2002.

D. DeCoste. Collaborative prediction using ensembles of maximum margin matrix factorizations. In *Proc. of ICML-06, 23rd Int. Conf. on Machine learning*, pages 249–256, Pittsburgh, Pennsylvania, USA, 2006.

N. Delannay and M. Verleysen. Collaborative filtering with interlaced generalized linear models. In *Proc. of ESANN-07, European Symp. on Artificial Neural Networks*, pages 247–252, Bruges, Belgium, 2007.

O. R. Duda, P. E. Hart., and D. G. Stork. *Pattern Classification*. John Wiley and Sons.

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.

K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: a constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001.

J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1):5–53, 2004.

W. Hill, L. Stead, M. Rosenstein, and G. Furnas. Recommending and evaluating choices in a virtual community of use. In *Proc. of CHI-95, ACM Conf. on Human Factors in Computing Systems*, pages 194–201, Denver, Colorado, USA, 1995. ISBN 0-201-84705-1.

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

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, California, USA, 2007.

D. D. Lee and H. S. Seung. Learning the parts of objects by nonnegative matrix factorization. *Nature*, 401(6755):788–791, 1999.

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, California, USA, 2007.

D. M. Pennock, E. Horvitz, S. Lawrence, and C. L. Giles. Collaborative filtering by personality diagnosis: a hybrid memory and model-based approach. In *Proc. of UAI-00, 16th Conf. on Uncertainty in Artificial Intelligence*, pages 473–480, Stanford, California, USA, 2000.

A. M. Rashid, S. K. Lam, G. Karypis, and J. Riedl. ClustKNN: a highly scalable hybrid model-& memory-based CF algorithm. In *Proc. of WebKDD-06, KDD Workshop on Web Mining and Web Usage Analysis, at 12th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, Philadelphia, Pennsylvania, USA, 2006.

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, 4th ACM Conf. on Computer Supported Cooperative Work*, pages 175–186, Chapel Hill, North Carolina, USA, 1994. ISBN 0-89791-689-1.

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*. MIT Press, Cambridge, Massachusetts, USA, 2008.

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

B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl. Application of dimensionality reduction in recommender system—a case study. In *Proc. of WebKDD-00, Web Mining for E-Commerce Workshop, at 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, Boston, Massachusetts, USA, 2000.

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.

N. Srebro and T. Jaakkola. Weighted low-rank approximations. In *Proc. of ICDM-03, 20th Int. Conf. on Machine Learning*, pages 720–727, Melbourne, Florida, USA, 2003.

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

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, California, USA, 2007.

G. Takács, I. Pilászy, B. Németh, and D. Tikk. A unified approach of factor models and neighbor based methods for large recommender systems. In *Proc. of ICADIWT-08, 1st IEEE Workshop on Recommender Systems and Personalized Retrieval*, pages 186–191, August 2008a.

G. Takács, I. Pilászy, B. Németh, and D. Tikk. Matrix factorization and neighbor based algoritms for the Netflix Prize problem. In *Proc. of RecSys-08, ACM Conf. on Recomender Systems*, pages 267–274, Lausanne, Switzerland, 2008b.

M. G. Vozalis and K. G. Margaritis. Using SVD and demographic data for the enhancement of generalized collaborative filtering. *Information Sciences*, 177(15):3017–3037, 2007.