Recommending New Movies: Even a Few Ratings Are More Valuable Than Metadata

István Pilászy [*]
Dept. of Measurement and Information Systems
Budapest University of Technology and Economics
Magyar Tudósok krt. 2.
Budapest, Hungary
pila@mit.bme.hu

Domonkos Tikk [*,†]
Dept. of Telecom. and Media Informatics
Budapest University of Technology and Economics
Magyar Tudósok krt. 2.
Budapest, Hungary
tikk@tmit.bme.hu

ABSTRACT

The Netflix Prize (NP) competition gave much attention to collaborative filtering (CF) approaches. Matrix factorization (MF) based CF approaches assign low dimensional feature vectors to users and items. We link CF and content-based filtering (CBF) by finding a linear transformation that transforms user or item descriptions so that they are as close as possible to the feature vectors generated by MF for CF.

We propose methods for explicit feedback that are able to handle 140 000 features when feature vectors are very sparse. With movie metadata collected for the NP movies we show that the prediction performance of the methods is comparable to that of CF, and can be used to predict user preferences on new movies.

We also investigate the value of movie metadata compared to movie ratings in regards of predictive power. We compare our solely CBF approach with a simple baseline rating-based predictor. We show that even 10 ratings of a new movie are more valuable than its metadata for predicting user ratings.

Categories and Subject Descriptors

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

General Terms

Algorithms, Experimentation

Keywords

content-based filtering, collaborative filtering, matrix factorization, NSVD1, Netflix Prize, RMSE

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

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

1. INTRODUCTION

The goal of recommender systems is to give personalized recommendation on items to users. Typically the recommendation is based on the former and current activity of the users, and metadata about users and items, if available.

There are two basic strategies that can be applied when generating recommendations. Collaborative filtering (CF) methods are based only on the activity of users, while content-based filtering (CBF) methods use only metadata. In this paper we propose hybrid methods, which try to benefit from both information sources.

The two most important families of CF methods are matrix factorization (MF) and neighbor-based approaches. Usually, the goal of MF is to find a low dimensional representation for both users and movies, i.e. each user and movie is associated with a feature vector. Movie metadata (which are mostly textual) can also be represented as a vector, using the usual vector-space model of text mining.

Our approach to connect CF and CBF methods works via the analog representation form. We aim to find a linear transformation that transforms the vectorial representation of movie metadata into the feature vector of the movie. This transformation can serve as a bridge between CF and CBF. We also propose generalizations to this basic method.

Content-based methods are essential when there is no available rating information on items, which is the case when new items are added into product lists. We investigate on a movie recommendation example how efficient predictors are metadata-based methods compared to rating-based ones. We point out that even if very few ratings are available, simple rating-based predictors outperform purely metadata-based ones.

We performed our experiments on the Netflix Prize (NP) dataset. This is currently the largest available CF dataset provided for the scientific community by Netflix. The NP dataset consists of 100 480 507 ratings of 480 189 users on 17 770 movies. Ratings are integer numbers on a 1-to-5 scale. In addition to that, the date of the ratings and the title and release year of the movies are also provided. We collected movie metadata from the internet for content-based filtering.

We also report on the time efficiency of our algorithms and show that the running time of the proposed methods ranges between a few minutes and an hour.

This paper is organized as follows. Section 2 introduces notations. Section 3 describes in detail some methods that form the basis of our proposed methods. Section 4 introduces our basic approach for CBF, which is then generalized in Section 5. Section 6 surveys related works on content-based filtering, and finally Section 7 evaluates the proposed methods on the NP-dataset, and this also includes the comparison of CBF and CF approaches.

2. NOTATION

Throughout this paper we use the following notations:

N: number of users

M: number of movies or items.

$u, v \in \{1, \ldots, N\}$: indices for users.

$i, j \in \{1, \ldots, M\}$: indices for movies or items.

r_{ui}: the rating of user u on item i.

$\hat{r}_{ui}$: the prediction of r_{ui}. In general, superscript "hat" denotes the prediction of the given quantity, that is, $\hat{x}$ is the prediction of x.

$\mathbf{R}$: the matrix of r_{ui} values.

$\mathcal{R}$: the set of (u, i) indices of $\mathbf{R}$ where ratings are provided;

n_u: number of ratings of user u, i.e. $n_u = |\{i : (u, i) \in \mathcal{R}\}|$.

$\mathbf{I}$: denotes the identity matrix of the appropriate size.

λ: regularization parameter.

η: learning rate for gradient methods.

C: dimension of the vector space of the vectorial representation of user or movie (item) metadata (depends on the context). Vectors in that space are usually sparse.

C', C'': like C, but for users and movies (items), respectively.

l: index for the metadata features ($l \in \{1, \ldots, C\}$ or $l \in \{1, \ldots, C'\}$ or $l \in \{1, \ldots, C''\}$).

3. BASIC ALGORITHMS

In this section we survey some factorization methods proposed originally for the Netflix Prize problem, which we will use in our further investigations. We assume that methods strive to minimize the prediction error in terms of RMSE on a validation set $\mathcal{V}$, which is computed as:

$$\text{RMSE} = \sqrt{\sum_{(u,i)\in\mathcal{V}} (r_{ui} - \hat{r}_{ui})^2 / |\mathcal{V}|},$$

Matrix factorization approaches have been applied successfully for both rating-based and implicit feedback-based CF problems [3, 4, 5, 6, 8, 9, 11]. The goal of MF methods is to approximate the matrix $\mathbf{R} \in \mathbb{R}^{N\times M}$ as a product of two lower rank matrices: $\mathbf{R} \approx \mathbf{P}\mathbf{Q}^{\mathrm{T}}$, where $\mathbf{P} \in \mathbb{R}^{N\times K}$ is the user feature matrix, $\mathbf{Q} \in \mathbb{R}^{M\times K}$ is the item (or movie) feature matrix, K is the number of features that is a predefined constant, and the approximation is only performed at $(u, i) \in \mathcal{R}$ positions. The r_{ui} element of $\mathbf{R}$ is approximated by

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

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

$$(\mathbf{P}^*, \mathbf{Q}^*) = \operatorname*{arg\,min}_{\mathbf{P},\mathbf{Q}} \sum_{(u,i)\in\mathcal{R}} \left(e_{ui}^2 + \lambda \mathbf{p}_u^{\mathrm{T}} \mathbf{p}_u + \lambda \mathbf{q}_i^{\mathrm{T}} \mathbf{q}_i \right). \quad (1)$$

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

In the NP competition two different algorithms were found to be very effective to approximately solve the above optimization problem: one based on Alternating Least Squares (ALS), proposed by team BellKor [3] and the other based on incremental gradient descent (also know as stochastic gradient descent), namely the Biased Regularized Incremental Simultaneous MF (BRISMF) proposed by team Gravity [9].

BellKor's alternating least squares approach alternates between two steps: step 1 fixes $\mathbf{P}$ and recomputes $\mathbf{Q}$, step 2 fixes $\mathbf{Q}$ and recomputes $\mathbf{P}$. The recomputation of $\mathbf{P}$ is performed by solving a separate least squares problem for each user: for the u-th user it takes the feature vector ($\mathbf{q}_i$) of movies rated by the user as input variables, and the value of the ratings (r_{ui}) as output variables, and finds the optimal $\mathbf{p}_u$ by ridge regression.

Borrowing the notations from [3], let the matrix $\mathbf{Q}[u] \in \mathbb{R}^{n_u\times K}$ denote the restriction of $\mathbf{Q}$ to the movies rated by user u, the vector $\mathbf{r}_u \in \mathbb{R}^{n_u\times 1}$ denote the ratings given by the u-th user to the corresponding movies, and let

$$\mathbf{A}_u = \mathbf{Q}[u]^{\mathrm{T}} \mathbf{Q}[u] = \sum_{i:(u,i)\in\mathcal{R}} \mathbf{q}_i \mathbf{q}_i^{\mathrm{T}}, \quad (2)$$

$$\mathbf{d}_u = \mathbf{Q}[u]^{\mathrm{T}} \mathbf{r}_u = \sum_{i:(u,i)\in\mathcal{R}} r_{ui} \cdot \mathbf{q}_i. \quad (3)$$

Then the ridge regression recomputes $\mathbf{p}_u$ as

$$\mathbf{p}_u = (\lambda n_u \mathbf{I} + \mathbf{A}_u)^{-1} \mathbf{d}_u. \quad (4)$$

According to [3, 5], the number of recomputations needed ranges between 10 and a "few tens".

Gravity's BRISMF approach works as follows [9]: the dataset is first ordered by user id, and then by rating date. The update of the model is performed per-rating, not per-user or per-movie. Suppose that we are at the (u, i)-th rating of $\mathbf{R}$. Then compute $e_{ui} = r_{ui} - \hat{r}_{ui}$, the error of the prediction. The update formulae for $\mathbf{p}_u$ and $\mathbf{q}_i$ are the followings:

$$\mathbf{p}'_u = \mathbf{p}_u + \eta \cdot (e_{ui} \cdot \mathbf{q}_i - \lambda \cdot \mathbf{p}_u), \quad (5)$$

$$\mathbf{q}'_i = \mathbf{q}_i + \eta \cdot (e_{ui} \cdot \mathbf{p}_u - \lambda \cdot \mathbf{q}_i) \quad (6)$$

One iteration over the database requires $O(|\mathcal{R}| \cdot K)$ steps, and the required number of iterations ranges between 1 and 14 [11]. It has been pointed out that larger K yields more accurate predictions [5, 8, 11].

Paterek introduced another factorization approach, called NSVD1 [6], where a prediction is made by the following rule:

$$\hat{r}_{ui} = b_u + c_i + {\mathbf{p}_u}^{\mathrm{T}} \mathbf{q}_i \quad (7)$$

Where:

$$\mathbf{p}_u = \left(\frac{1}{\sqrt{n_u}} \sum_{j:\ (u,j)\in\mathcal{R}} \mathbf{w}_j \right) \quad (8)$$

Here the user feature vector $\mathbf{p}_u$ is a function of other variables, as opposed to MF, where it is "arbitrary". The variables b_u and c_i are the user and movie biases. The $\mathbf{w}_j$ vectors are the secondary movie feature vectors, i.e. NSVD1 has two sets of movie feature vectors. Paterek did not provide a training algorithm for NSVD1, we will apply the NSVD1 training algorithm of [9], which we will describe in the next section.

4. USING NSVD1 FOR CONTENT-BASED FILTERING

In this section we propose a pure movie-metadata-based CBF framework.

Paterek's NSVD1 algorithm can be interpreted in the following way: first, users are represented by M dimensional binary vectors. Such a vector reflects which movies are rated by a user and which not, and is used to infer $\mathbf{p}_u$ by a linear transformation, denoted by the matrix $\mathbf{W}$. As opposed to a learning algorithm for MF that finds $\mathbf{P}$ and $\mathbf{Q}$, a learning algorithm for NSVD1 finds $\mathbf{Q}$ and $\mathbf{W}$. Users can also be represented by vectors derived not only from their ratings but also from their metadata (demographic data). From the viewpoint of a learning algorithm, there is no difference between a vector describing user metadata and a vector containing 1 at the movies rated by the user. In other words, we can consider the list of movie IDs rated by the user as a textual description about the user.

One can run the NSVD1 algorithm interchanging the role of users and movies. Then movies are represented by vectors indicating which users rated them. For clarity, we refer to the original version as user-NSVD1, and to its dual as movie-NSVD1.

Algorithm 1 is the training algorithm for movie-NSVD1, based on [9], using the following notations:

- $\mathbf{W} \in \mathbb{R}^{C\times K}$ is the transformation matrix that transforms movie metadata to feature vectors.
- $\mathbf{w}_l \in \mathbb{R}^{K\times 1}$ is the transpose of the l-th row of $\mathbf{W}$.
- $\mathbf{X} \in \mathbb{R}^{N\times C}$ contains metadata about the movies; note that $\mathbf{X}$ is fully specified as opposed to $\mathbf{R}$, since we may assume that movie metadata are correct and complete, while most elements of $\mathbf{R}$ are unknown.
- $\mathbf{x}_i \in \mathbb{R}^{C\times 1}$ is the transpose of the i-th row of $\mathbf{X}$.
- x_{il} is the (i,l)-th element of $\mathbf{X}$.

With this notation:

$$\mathbf{q}_i = \mathbf{W}^{\mathrm{T}}\mathbf{x}_i = \sum_{l:\ x_{il}\neq 0} x_{il}\cdot \mathbf{w}_l, \tag{9}$$

$$\mathbf{Q} = \mathbf{X}\mathbf{W}, \tag{10}$$

$$\hat{\mathbf{R}} = \mathbf{P}\mathbf{Q}^{\mathrm{T}} = \mathbf{P}(\mathbf{X}\mathbf{W})^{\mathrm{T}}, \tag{11}$$

where $\hat{\mathbf{R}}$ is the approximation (prediction) of $\mathbf{R}$.

Line 5 of Algorithm 1 computes the movie feature vector. Lines 8–14 updates the movie feature vector (together with the user feature vector and the two biases), in the same way as in BRISMF. Line 16 is new in this algorithm compared to the NSVD1 algorithm in [9], it allows x_{il} to be arbitrary; in the original version x_{il} is set to $1/\sqrt{n_i}$ for $(l,i)\in\mathcal{R}$, and 0 otherwise, thus, for any i, $\sum_l x_{il}^2 = 1$.

Lines 17–19 backpropagate the change in $\mathbf{q}_i$ to $\mathbf{W}$. After this step, $\mathbf{q}_i$ is equal to $\mathbf{W}^{\mathrm{T}}\mathbf{x}_i$, i.e. as defined in line 5. To prove it, we rewrite that 3 lines in the following form:

$$\mathbf{W} \leftarrow \mathbf{W} + \frac{\mathbf{x}_i}{\mathbf{x}_i{}^{\mathrm{T}}\mathbf{x}_i}(\mathbf{q}_i - \mathbf{a})^{\mathrm{T}}, \tag{12}$$

and check what line 5 yields with the new $\mathbf{W}$:

$$\left(\mathbf{W} + \frac{\mathbf{x}_i}{\mathbf{x}_i{}^{\mathrm{T}}\mathbf{x}_i}(\mathbf{q}_i - \mathbf{a})^{\mathrm{T}}\right)^{\mathrm{T}}\mathbf{x}_i =$$

$$= \left(\mathbf{W}^{\mathrm{T}} + (\mathbf{q}_i - \mathbf{a})\frac{\mathbf{x}_i{}^{\mathrm{T}}}{\mathbf{x}_i{}^{\mathrm{T}}\mathbf{x}_i}\right)\mathbf{x}_i = \mathbf{W}^{\mathrm{T}}\mathbf{x}_i + (\mathbf{q}_i - \mathbf{a}).$$

Since $\mathbf{a}$ is $\mathbf{W}^{\mathrm{T}}\mathbf{x}_i$ by definition, they are equal.

```
1  Create random model.
2  repeat
3      one epoch:
4      for $i \leftarrow 1$ to $M$ do
5          $\mathbf{q}_i \leftarrow \mathbf{W}^{\mathrm{T}}\mathbf{x}_i$
6          $\mathbf{a} \leftarrow \mathbf{q}_i$
7          foreach $u:\ (u,i)\in\mathcal{R}$ do
8              $\hat{r}_{ui} \leftarrow b_u + c_i + \mathbf{p}_u{}^{\mathrm{T}}\mathbf{q}_i$
9              $e_{ui} \leftarrow r_{ui} - \hat{r}_{ui}$
10             $\mathbf{b} \leftarrow \mathbf{p}_u$
11             $\mathbf{p}_u \leftarrow \mathbf{p}_u + \eta\cdot(e_{ui}\cdot\mathbf{q}_i - \lambda\cdot\mathbf{p}_u)$
12             $\mathbf{q}_i \leftarrow \mathbf{q}_i + \eta\cdot(e_{ui}\cdot\mathbf{b} - \lambda\cdot\mathbf{q}_i)$
13             $b_u \leftarrow b_u + \eta\cdot(e_{ui} - \lambda\cdot b_u)$
14             $c_i \leftarrow c_i + \eta\cdot(e_{ui} - \lambda\cdot c_i)$
15         end
16         $d \leftarrow 1/(\mathbf{x}_i{}^{\mathrm{T}}\mathbf{x}_i)$
17         foreach $l:\ x_{il}\neq 0$ do
18             $\mathbf{w}_l \leftarrow \mathbf{w}_l + d\cdot x_{il}\cdot(\mathbf{q}_i - \mathbf{a})$
19         end
20     end
21 until RMSE on the test set decreases ;
```

Algorithm 1: Training algorithm for movie-NSVD1

Let us analyze the computational complexity of Algorithm 1:

- computing $\mathbf{p}_u$: $O(K\cdot\sum_{l:\ x_{il}\neq 0} 1)$, executed for each i.
- lines 8–14: $O(K)$, executed for $|\mathcal{R}|$ times.
- line 16: $\sum_{l:\ x_{il}\neq 0} 1$, executed for each i.
- lines 17–19: $K\cdot\sum_{l:\ x_{il}\neq 0} 1$, executed for each i.

Let $\mathcal{X} = \{(i,l):\ x_{il}\neq 0\}$ denote the set of indices of non-zero elements of matrix $\mathbf{X}$. With this notation, the time requirement of one epoch is:

$$O(K\cdot|\mathcal{X}| + K\cdot|\mathcal{R}|). \tag{13}$$

The optimal number of epochs depends on the experiment.

Note that the time complexity is very favorable: if $|\mathcal{X}|$ is proportional to the size of the movie descriptions measured in bytes (this holds usually when most of the information is textual, and we apply the vector-space model to represent texts as vectors), then this complexity is equal to reading the movie descriptions and the ratings K-times.

Batch tranining for NSVD1.

We also propose a different training algorithm for NSVD1, which is described in Algorithm 2. It contains two important modifications compared to Algorithm 1:

- It updates $\mathbf{W}$ after iterating over all movies (at the end of each epoch), i.e. $\mathbf{W}$ is updated in batch mode.
- we introduce new parameters: learning rate η_2, regularization factor λ_2, and the iteration count n_2. If we set $\eta_2 = 1$, $\lambda_2 = 0$ and $n_2 = 1$, we get the special case of Algorithm 1, but in batch mode.

Algorithm 2 does not aim to perfectly appoximate $\mathbf{Q}$, unless $\eta_2 = 1$ and $\lambda_2 = 0$. To compensate this, the update process is repeated for n_2 times, to have a more accurate approximation of $\mathbf{Q}$.

The main advantage of this method will be clear in the next section:

- It will allow to alloy alternating least squares (ALS) and NSVD1.

- Using ALS, it can be applied to handle implicit feedback data efficiently, i.e. when $\mathbf{R}$ is fully filled.
- It enables the use of user and movie metadata simultaneously.

One epoch of Algorithm 1 has time complexity $O(K\cdot|\mathcal{X}|+K\cdot|\mathcal{R}|)$. Algorithm 2 updates the whole $\mathbf{W}$ for each i, unless $\lambda_2=0$, in which case the sparsity of $\mathbf{x}_i$ can be exploited by updating only those rows of $\mathbf{W}$, which corresponds to $x_{il}\neq 0$. However, we can apply a trick to speed up the process for the $\lambda_2\neq 0$ case as well: we can decompose $\mathbf{W}$ into $\mathbf{W}=v\cdot\mathbf{V}$. In the beginning, let $\mathbf{V}=\mathbf{W}$, and $v=1$. We can replace line 15 of Algorithm 2 by the following two equivalent lines:

$$\mathbf{W}\leftarrow(1-\eta_2\lambda_2)\mathbf{W} \tag{14}$$

$$\mathbf{W}\leftarrow\mathbf{W}+\eta_2\cdot(d\cdot\mathbf{x}_i\cdot\mathbf{e}^{\mathrm{T}}) \tag{15}$$

The first line does the regularization, the second decreases the training error. With the decomposition of $\mathbf{W}$ into $v\cdot\mathbf{V}$, we can rewrite these lines as:

$$v\leftarrow(1-\eta_2\lambda_2)\cdot v \tag{16}$$

$$\mathbf{V}\leftarrow\mathbf{V}+\eta_2\cdot(d\cdot\mathbf{x}_i\cdot\mathbf{e}^{\mathrm{T}})/v \tag{17}$$

With this trick, the sparsity of $\mathbf{x}_i$ can be exploited in the update of $\mathbf{V}$, thus the time complexity of Algorithm 2 with this modification is:

$$O(K\cdot n_2\cdot|\mathcal{X}|+K\cdot|\mathcal{R}|).$$

```
1  Create random model.
2  repeat
3      one epoch:
4      $\mathbf{Q}\leftarrow\mathbf{XW}$
5      foreach $(u,i)\in\mathcal{R}$ do
6          update $b_u$, $c_i$, $\mathbf{p}_u$ and $\mathbf{q}_i$ as in
7          Algorithm 1, lines 8–14.
8      end
9      // now update $\mathbf{W}$ such that the actual value of
10     // $\mathbf{Q}$ is better approximated by $\mathbf{XW}$
11     for 1 to $n_2$ do
12         for $i\leftarrow 1$ to $M$ do
13             $\mathbf{e}\leftarrow\mathbf{q}_i-\mathbf{W}^{\mathrm{T}}\mathbf{x}_i$
14             $d\leftarrow 1/(\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_i)$
15             $\mathbf{W}\leftarrow\mathbf{W}+\eta_2\cdot(d\cdot\mathbf{x}_i\cdot\mathbf{e}^{\mathrm{T}}-\lambda_2\cdot\mathbf{W})$
16         end
17     end
18     $\mathbf{Q}\leftarrow\mathbf{XW}$
19 until RMSE on the test set decreases ;
```

Algorithm 2: Batch training algorithm for movie-NSVD1

Algorithm 1 requires the training examples to be preordered by i, otherwise lines 17–19 would have to be executed more than once per movie. Algorithm 2 can process examples in arbitrary order, as it applies batch-backpropagation to update $\mathbf{W}$.

5. GENERALIZED APPROACH

Recall that a prediction for the (u,i)-th rating of factorization-based models is the following:

$$\hat{r}_{ui}=b_u+c_i+\mathbf{p}_u^{\mathrm{T}}\mathbf{q}_i \tag{18}$$

The idea is to generalize Algorithm 2 in two ways:

- allow using both user and movie metadata;
- not all user and/or item features depend on metadata.

In [9] a hybrid MF-NSVD1 approach was proposed, which contained a similar idea: part of the user features were arbitrary (the MF-part), and the rest of the user features were computed by a linear transformation from a binary vector indicating which movies were rated by the user (the NSVD1-part).

Let the user features with indices $k\in\{K_1,\ldots,K_2\}$ be computed by linear transformation from metadata. Here K_1 and K_2 are the first and last indices of such features. Similarly, let the item features with indices $k\in\{K_3,\ldots,K_4\}$ be dependenent on metadata, where K_3 and K_4 are the first and last indices of such features. Let $\mathbf{P}[1..N;K_1..K_2]$ denote the submatrix of $\mathbf{P}$, referring to rows $1\leq u\leq N$ and columns $K_1\leq k\leq K_2$. Similarly, let $\mathbf{Q}[1..M;K_3..K_4]$ denote the submatrix of $\mathbf{Q}$, referring to rows $1\leq i\leq M$ and columns $K_3\leq k\leq K_4$.

The learning algorithm for our general variant is described in Algorithm 3. Here $\mathbf{X}'\in\mathbb{R}^{N\times C'}$ and $\mathbf{X}''\in\mathbb{R}^{M\times C''}$ denote the matrix of user and movie metadata, resp. The matrices $\mathbf{W}'\in\mathbb{R}^{C'\times(K_2-K_1+1)}$ and $\mathbf{W}''\in\mathbb{R}^{C''\times(K_4-K_3+1)}$ are used to recompute the submatrices of $\mathbf{P}$ and $\mathbf{Q}$ from metadata.

```
1  Create random model.
2  repeat
3      one epoch:
4      $\mathbf{P}[1..N;K_1..K_2]\leftarrow\mathbf{X}'\mathbf{W}'$
5      $\mathbf{Q}[1..M;K_3..K_4]\leftarrow\mathbf{X}''\mathbf{W}''$
6      foreach $(u,i)\in\mathcal{R}$ do
7          update $b_u$, $c_i$, $\mathbf{p}_u$ and $\mathbf{q}_i$ as in
8          Algorithm 1, lines 8–14.
9      end
10     for 1 to $n_2$ do
11         for $u\leftarrow 1$ to $N$ do
12             $\mathbf{e}\leftarrow\mathbf{P}[u;K_1..K_2]^{\mathrm{T}}-\mathbf{W}'^{\mathrm{T}}\mathbf{x}'_u$
13             $d\leftarrow 1/(\mathbf{x}'^{\mathrm{T}}_u\mathbf{x}'_u)$
14             $\mathbf{W}'\leftarrow\mathbf{W}'+\eta_2\cdot(d\cdot\mathbf{x}'_u\cdot\mathbf{e}^{\mathrm{T}}-\lambda_2\cdot\mathbf{W}')$
15         end
16     end
17     $\mathbf{P}[1..N;K_1..K_2]\leftarrow\mathbf{X}'\mathbf{W}'$
18     for 1 to $n_2$ do
19         for $i\leftarrow 1$ to $M$ do
20             $\mathbf{e}\leftarrow\mathbf{Q}[i;K_3..K_4]^{\mathrm{T}}-\mathbf{W}''^{\mathrm{T}}\mathbf{x}''_i$
21             $d\leftarrow 1/(\mathbf{x}''^{\mathrm{T}}_i\mathbf{x}''_i)$
22             $\mathbf{W}''\leftarrow\mathbf{W}''+\eta_2\cdot(d\cdot\mathbf{x}''_i\cdot\mathbf{e}^{\mathrm{T}}-\lambda_2\cdot\mathbf{W}'')$
23         end
24     end
25     $\mathbf{Q}[1..M;K_3..K_4]\leftarrow\mathbf{X}''\mathbf{W}''$
26 until RMSE on the test set decreases ;
```

Algorithm 3: Batch training algorithm for a generalized NSVD1

5.1 Learning methods

Although Algorithm 3 updates $\mathbf{W}'$ and $\mathbf{W}''$ at the end of the epochs (batch updating), there are also another possibilities, for example, we can update $\mathbf{W}'$ at the end of the epochs (as in Algorithm 2), and update $\mathbf{W}''$ after processing all ratings of an item (as in Algorithm 1) and before turning to the next item. We propose to use the following three learning methods for modifying b_u, c_i, $\mathbf{p}_u$ and $\mathbf{q}_i$, to decrease e_{ui}:

- IGD – incremental gradient descent: this is described in Algorithm 1, lines 8–14.
- AGD – alternating incremental gradient descent: in even epochs, only b_u and $\mathbf{p}_u$ are modified. In odd epochs, only c_i and $\mathbf{q}_i$ are modified (or vice versa).
- ALS – alternating least squares [3]. In even epochs, it recomputes $\mathbf{P}$, in odd epochs it recomputes $\mathbf{Q}$ (or vice versa) with a least squares method. We can incorporate biases b_u and c_i, by fixing a particular column of $\mathbf{P}$ and $\mathbf{Q}$ to have the constant value 1, as it has been suggested in [11].

To update $\mathbf{W}'$ or $\mathbf{W}''$, we can apply incremental or batch method, independently on user side and on movie side, however, there is one exception: incremental updates on both side, with IGD, since user-incremental updating requires user-ordering of the ratings. Moreover, in case of ALS the incremental updates makes no sense, since the user or movie features are completely recomputed, not updated.

Advantages of AGD: if we have both user and movie metadata, we can apply incremental backpropagation for both $\mathbf{X}''$ and $\mathbf{X}'$, using user-ordered data in even epochs, movie-ordered data in odd epochs. Note that AGD allows parallelization (like ALS), since the order how we process users (or movies) does not change the results.

With ALS, we can get NSVD1-like training for implicit feedback datasets (currently only ALS has an efficient method to cope with the fully filled matrix, see [5]).

Using batch training and only movie metadata, one can consider the ALS approach in the following way: using $\mathbf{R} = \mathbf{P}(\mathbf{X}\mathbf{W})^{\mathrm{T}}$, in even epochs, $\mathbf{P}$ is estimated, in odd epochs, first $\mathbf{Q} = \mathbf{X}\mathbf{W}$ is estimated, and then using $\mathbf{Q} = \mathbf{X}\mathbf{W}$, only $\mathbf{W}$ is estimated.

6. RELATED WORK

There are many works on CBF and hybrid CF-CBF systems. Our methods are built on new methods that emerged in the Netflix Prize competition, namely on MF with ALS or incremental gradient descent, and NSVD1. We give a briefly survey on some previous articles: Basilico et al. suggested the following formula for prediction (rewritten for better readability, [1]):

$$\hat{r}_{ui} = \sum_{(v,j)\in\mathcal{R}} \alpha_{vj} \cdot \mathrm{sim}^{(1)}_{uv} \cdot \mathrm{sim}^{(2)}_{ij}, \qquad (19)$$

where $\mathrm{sim}^{(1)}_{uv}$ and $\mathrm{sim}^{(2)}_{ij}$ are the predefined user and movie similarity measures that tell how similar are the metadata of two users or items; here α_{vj} is to be optimized. One prediction requires $O(|\mathcal{R}|)$ time, which makes this approach unpractical.

Zhang et al. proposed an interesting approach for CBF [12] which is somewhat similar to ours: in their approach, $\mathbf{Q}$ is filled with movie metadata (i.e. $\mathbf{Q} = \mathbf{X}''$), and using the $\mathbf{R} = \mathbf{P}\mathbf{Q}^{\mathrm{T}}$ formula, $\mathbf{P}$ is estimated by least squares solver, however, the covariance matrix is modified with a matrix $\mathbf{\Sigma}$. Their method alternates between recomputing $\mathbf{P}$ and $\mathbf{\Sigma}$. Both their method and ours aim to intensify (or suppress) in a collaborative way those patterns in the metadata that can be useful (or useless). For example: the term "Season" can have large impact on $\hat{r}_{ui}$ by either giving it high weight in the movie or in the user feature vectors. If it is a good feature (many users rely on that feature), it should get high weight on the movie side (allowing lower norm of $\mathbf{p}_u$ in general). On the other hand, if it is a bad feature (few users rely on that feature), it should be given a low weight, allowing more important movie features to have higher weight. They concluded that the method was slow, thus they proposed a diagonal $\mathbf{\Sigma}$ that could handle the NP dataset in 4 hours. They reported results only on a small subset of the NP dataset, which makes that incomparable to ours.

Collective matrix factorization.

Singh et al. proposed a general approach to solve multiple matrix factorization tasks simultaneously [7], called Collective Matrix Factorization (CMF). We briefly summarize their idea: suppose that we have two matrices (there may be missing values), one for the user-movie relation, describing how users rate movies ($\mathbf{R}$), the other for the movie-metadata relation ($\mathbf{X}''$). We can factorize both matrices separately, and get movie feature vectors from factorizing $\mathbf{R}$, and get different movie feature vectors from factorizing $\mathbf{X}''$. The idea is to share the movie feature vectors (no difference), and formulate a new optimization problem by linearly combining the target function of the two factorizations (cf. eq. (1)), with coefficients α_1 and α_2. The idea can be easily generalized to factorize any number of matrices simultaneously.

For content-based filtering, we are interested only in the movie-metadata and user-metadata relations. One can think at Singh's optimization problem in the following, more intuitive way: we concatenate $\mathbf{R} \in \mathbb{R}^{N\times M}$ and $\mathbf{X}''^{\mathrm{T}} \in \mathbb{R}^{C''\times M}$, and have a new $\mathbf{R}' \in \mathbb{R}^{(N+C'')\times M}$ matrix of ratings and "pseudo-ratings". This matrix is then to be factorized. Suppose that one column of $\mathbf{X}''$ is 1 iff the term "Season" occurs in the movie title, and 0 otherwise. In $\mathbf{R}'$, we have a corresponding pseudo-user, who rated every movie having "Season" in the title 1, and all other movies 0. When a new movie gets into the recommender system, no real ratings are available, but we have pseudo-ratings. As more ratings become available, the less the impact of pseudo-ratings is. Two issues have to be solved: handling α_1 and α_2, and the effective factorization of $\mathbf{R}'$. Since $\mathbf{R}'$ can contain many ratings (e.g. when $C'' = 60000$ and $M = 17770$, it is 10 times larger than the Netflix Prize dataset), incremental gradient descent may be slow. Hu et al. proposed an efficient weighted ALS algorithm to handle implicit feedback datasets [5], i.e. when the matrix is fully filled (mostly with zeros) and an importance weight is assigned to each element. To factorize $\mathbf{R}'$ with that method, we set the weight of $(u,i) \notin \mathcal{R}$ examples to 0, the weight of $(u,i) \in \mathcal{R}$ to α_1, and the rest of the weights (which corresponds to the metadata part) to α_2.

Both user and movie metadata may be incorporated, by extending $\mathbf{R}'$ with C' new columns, corresponding to user metadata. The lower right $C'' \times C'$ block of the new $\mathbf{R}''$ is set to zero.

7. EXPERIMENTS

7.1 The Netflix Prize Dataset

We evaluated our algorithm against the Netflix Prize dataset. Netflix provides a train-test split of the ratings. The test set contains 1 408 395 ratings and is called Probe. Noteworthy that Probe contains newer ratings for each user than the Train. In our experiments we kept only a fixed 1/10 subset of the Probe ratings (140 840 ratings), and put the rest of Probe into Train. This new test set is termed as "Probe10".[1] Thus, we train using $100\,480\,507 - 140\,840$ ratings, and evaluate on 140 840 ratings.

Netflix has also released another evaluation set, which is termed as Quiz. It contains cca. 2 817 131/2 ratings, but r_{ui}-s are withheld. The goal of the competition is to predict those r_{ui} values. We observed that evaluating on Probe10 yields similar results in terms of RMSE as evaluating on Quiz [9]. The difference is almost always less than 0.0002. Unless we explicitly mention, from now on the RMSE values refer to the Probe10 RMSE.

We collected movie metadata from `www.netflix.com` for all but 106 movies out of the 17 770. We considered all metadata as text, and used the vector space model common in text-mining to represent texts as vectors. We distinguished between word occurrences in different zones by zone-prefixing. That is the word "comedy" occurring in the genre and in the title zones are represented in two different dimensions "genre:comedy" and "title:comedy" of the metadata vector space. We used the following zones: title, synopsis, actor, director, release year, genre. We did not split the name of the actors and directors into first and last name.

Altogether, movies are described with $C'' = 146\,810$ dimensional vectors. These vectors have 81 nonzero values on average. Movie vectors were normalized.

Before experimentation, we subtracted the global mean, 3.6043, from all ratings. The prediction of the algorithms were linearly transformed to have the same mean and standard deviation as the Probe10 set.

All experiments were performed on an Intel Core2 Quad Q9300 cpu on 3.3GHz, using only 1 core.

7.2 Results

Basic predictors.

First, we experimented with some MF methods, using no metadata:

- IGD01: matrix factorization (MF) with biases b_u and c_i, using incremental gradient descent. Using the following parameters: $\eta = \lambda = 0.005$, $K = 20$. Best Probe10 RMSE in the 15th epoch: 0.9079. Running time: 304 sec.
- AGD01: like IGD01, but with alternating gradient descent. Best Probe10 RMSE in the 28th epoch: 0.9096. Running time: 675 sec.
- ALS01: MF with biases b_u and c_i, using alternating least squares, starting with movie features. Using the following parameters: $K = 10$, $\lambda^{(p)} = 2$, $\lambda^{(q)} = 0.001$. Probe10 RMSE in some epochs: 10 : 0.9305, 20 : 0.9264, 30 : 0.9259, 40 : 0.9257. Running time for 40 epochs: 1431 sec.

The main contribution is that AGD01 converges, although its performance is not as good as of IGD01. Interestingly, it requires cca. twice as many epochs as IGD01, that is, almost the same number of user and movie feature updates.

[1]A Perl script is available at our homepage, `gravityrd.com`, which selects the Probe10 from the original Netflix Probe set to ensure repeatability.

NSVD1 with movie metadata.

We now evaluate the movie-NSVD1 proposed in Section 4. We experimented with various subsets of movie metadata, which are denoted by:

- T1: keeping all features. $C'' = 146\,810$; 81 nonzero features on average,
- T2: keeping only those features that occur in at least 2 movies. $C'' = 64\,746$; 76 nonzero features on average.
- T4: like T2, but at least 4 movies. $C'' = 33\,838$; 72 nonzero features on average.
- T10: like T2, but at least 10 movies. $C'' = 14\,547$; 66 nonzero features on average.
- T20: like T2, but at least 20 movies. $C'' = 7\,340$; 60 nonzero features on average.

We used the same learning parameters as in IGD01 or AGD01, but now all movie features are computed by a linear transformation from their descriptions. We experimented with 4 different variants:

- IGD-I: IGD, with incremental backpropagation. Using movie-ordered database.
- AGD-I: AGD, with incremental backpropagation.
- IGD-B: IGD, with batch-backpropagation. Using (user, time)-ordered database.
- AGD-B: AGD, with batch-backpropagation.

For B variants, we experimented with $n_2 = 1$ and $n_2 = 10$.

Table 1 summarizes the results of experiments. The optimal number of epochs varied between 24–31 for AGD-I and AGD-B, and 11–15 for IGD-I and IGD-B. One epoch required 40 seconds for IGD-I and AGD-I on average.

Batch variants gave better Probe10 RMSE, when $n_2 = 10$. When we set n_2 to 1, Probe10 RMSE got worse. IGD-B can be better than IGD-I due to another reason: it allows to use the (user,date)-ordered training set (no speed problems), which was found to be beneficial [10] for incremental gradient descent.

	n_2	T1	T2	T4	T10	T20
IGD-I	–	0.9143	0.9155	0.9161	0.9200	0.9285
IGD-B	1	0.9121	0.9130	0.9147	0.9219	0.9370
IGD-B	10	0.9089	0.9093	0.9101	0.9157	0.9322
AGD-I	–	0.9165	0.9175	0.9186	0.9224	0.9306
AGD-B	1	0.9148	0.9163	0.9184	0.9265	0.9401
AGD-B	10	0.9112	0.9117	0.9120	0.9170	0.9312
ALS-B	1	0.9429	0.9461	0.9509	0.9616	0.9706
ALS-B	10	0.9327	0.9350	0.9389	0.9608	0.9716

Table 1: Probe10 RMSE of movie-NSVD1 with different metadata and learning methods

Clearly, more metadata imply better prediction performance. However, this is obvious: consider a simple example, where each movie has only one unique metadata, that is, $C'' = M$, and $\mathbf{X}''$ is the identity matrix. This case falls back to the matrix factorization, since the movie features have no inter-dependency, As we add infrequent metadata (i.e. words that occur only in one or few movie descriptions), the movie features have more freedom (less inter-dependency between movies), as movie descriptions tend to be dissimilar, thus the results should get closer to the results of pure matrix factorization, i.e. to the RMSE of the IGD01, AGD01 and ALS01 methods (0.9079, 0.9096 and 0.9305).

Predicting ratings on new movies.

CBF has a great advantage over CF: the ability to handle new movies. When a new movie is added to a recommendation system, it has no ratings, but metadata are usually available. In this case, CBF algorithms may help. We created the following setup to measure the capability of some CBF algorithms at handling new movies: first, we created 10 partitions for movies, and then applied 10-fold cross-validation: we trained a model using only the appropriate 9/10 of the original training data (i.e. all ratings except Probe10), and then evaluated the model on the appropriate 1/10 of the original test data (i.e. Probe10). Thus, one evaluation means training on $\sim 0.9 \cdot (100\,480\,507 - 140\,840)$ ratings, and evaluating on $\sim 0.1 \cdot 140\,840$ ratings. Final Probe10 RMSE was calculated by adding the sum of squared errors of each of the 10 evaluations. We refer to this final Probe10 RMSE as **X10 RMSE**.

If the descriptions of movies are as valuable as their ratings, then X10 RMSE should be equal to the Probe10 RMSE of the previous experiments where movies has many ratings. On the other hand, if it is unhelpful in predicting user preferences, then X10 RMSE should be not much better than that of a user-bias-based method (where the prediction is based on the average rating of the active user).

First, we define some basic predictors:

- user-average predictor: $\hat{r}_{ui} = b_u$, where $b_u = \sum_{i:(u,i)\in\mathcal{R}} r_{ui} / (|\{i : (u,i) \in \mathcal{R}\}| + \alpha_1)$.
- bias predictor: $\hat{r}_{ui} = b_u + c_i$, where b_u and c_i are estimated as the global effects in [2]. However, we run the estimation process more than once, thus, both b_u and c_i will be re-estimated. For new items, $c_i = 0$. Let α_1 and α_2 denote the α used to reestimate b_u and c_i, resp.
- user-bias predictor with incremental gradient method: $\hat{r}_{ui} = b_u$ again, but we use incremental gradient method to give larger weights to newer examples.
- input-bias predictor: $\hat{r}_{ui} = b_u + \mathbf{w}^{\mathrm{T}}\mathbf{x}_i$, where b_u and $\mathbf{w}$ are estimated using ALS. b_u and $\mathbf{w}$ are regularized with α_1 and α_3, resp. On can think at the input-bias predictor as the movie bias c_i is estimated by a linear function of the descriptions of the movies.

We did many experiments to optimize α_1, α_2 and α_3.

For user-average predictor, X10 RMSE = 1.0616 ($\alpha_1 = 2$).

Interestingly, for the bias predictor, $\alpha_1 = 2$ and $\alpha_2 = 5 \cdot 10^5$ yielded the best results: X10 RMSE = 1.0615. Larger values (e.g. $\alpha_2 = 2 \cdot 10^8$) increased RMSE only by 1/3 of 0.0001, while smaller values (e.g. $\alpha_2 = 100$) increased it by 0.0038, even smaller α_2 values were even worse. We interpret this as modeling movie biases does not decrease the X10 RMSE.

For the user-bias predictor with incremental gradient method, $\eta = 0.05$ and $\lambda = 0.005$ gave X10 RMSE = 1.0504.

For the input-bias predictor, we tried all the T1, ..., T20 datasets. T1 performed the best at $\alpha_1 = 5$ and $\alpha_3 = 5000$ and yielded X10 RMSE = 1.0305. Changing the weighting scheme of the metadata vector (e.g. to tf-idf) did not improve the results.

Next, we examined the prediction performance of the proposed movie-NSVD1 for CBF without movie biases using T4. We fine-tuned the learning parameters for NSVD1 with $K = 30$ using the parameter optimizer of [10], optimizing only on the first tenth of the 10 folds. We stopped the parameter tuning process after 127 runs. The model with the best X10 RMSE (1.0080) was built in 10 epochs, requiring 366s in total; the 10-fold cross validation thus required about 1 hour. We submitted this predictor to Netflix in order to test the possible overlearning of the parameter optimization. As Quiz RMSE was 1.0078, we could exclude this side-effect.

Then we tried the T1, ..., T20 datasets. T2 performed best, using tf-idf term weighting. Increasing K yielded better results, with $K = 500$, we obtained X10 RMSE = 0.9990. The running time was 2570 seconds.

Clearly, NSVD1 has better RMSE than the other methods. However, the Probe10 RMSE (not the cross-validated) of a simple bias predictor is far better than 1.0080: it is usually between 0.9700 and 0.9900. This means, that when we know the movie average, c_i, it is a much more valuable piece of information than any metadata (the title, genre, etc. of movies). Next we investigate how many ratings a movie should have to obtain a reliable enough estimate of c_i.

NSVD1: ratings are more valuable than metadata.

We compared the following two methods: NSVD1 with the cross-validation procedure mentioned above (1/10 of the movies are skipped), and the bias predictor (using both user-bias and movie-bias) with incremental gradient method, where the first N ratings of the skipped movies were included in the training set. We varied N, and checked when the RMSE of such a simple predictor will pass over the RMSE of the movie-NSVD1. Results are summarized in Table 2. When all ratings are used, Probe10 RMSE is 0.9721.

One can observe in Table 2 that even $N = 10$ ratings are more valuable than the metadata. Recall that this comparison was performed between the very simple bias predictor evaluated on not-so-new movies and the optimized movie-NSVD1 evaluated on new movies. Most probably, having a more sophisticated rating-based method would yield an even larger gap between the results of the two approaches.

N	0	1	2	5	10	20
RMSE	1.0564	1.0447	1.0325	1.0131	0.9978	0.9880
N	30	40	50	100	200	500
RMSE	0.9841	0.9819	0.9802	0.9772	0.9754	0.9740

Table 2: X10 RMSE of bias predictors, when N ratings from each skipped movie are added to the training set

CMF with movie metadata, actor similarity.

We implemented the CMF method of [7]. We were unable to get X10 RMSE results below 1.0800, however the algorithm was fast (400 seconds for training with $K = 10$).

Then we focused on the models. The factorization of the matrix of pseudo-ratings gives interesting results, compared to the $\mathbf{W}''$ of movie-NSVD1. Each metadata element is associated with a K-dimensional feature vector in both approaches ($\mathbf{w}_l$ in the movie-NSVD1 approach). Let $\mathbf{W}''' \in \mathbb{R}^{C'' \times K}$ denote the feature vectors of CMF for movie metadata.

In order to compare the two approaches, we investigated how effective are the models in finding similar actors. Given an actor, we can specify a ranked list of similar actors via the feature vectors of movie metadata using an appropriate similarity measure (we used S2 of [10]). When querying for *Bruce Willis* we obtained as top 3 similar actors:

- NSVD1: Mel Gibson, Julia Roberts, Tom Hanks
- CMF: Robert De Niro, Nicolas Cage, Keanu Reeves

The top 10 list of NSVD1 contains also one director and two synopsis words, while CMF returns only actors. For *Jennifer Lopez* we obtained

- NSVD1: Ben Affleck, Sandra Bullock, Julia Roberts
- CMF: Sandra Bullock, Drew Barrymore, Julia Roberts

Although the results are similar, NSVD1 returned one male actor beside the two females. We examined a couple of other actors, and found that CMF performed consistently better, i.e., it yielded only actors for each queried actor and did not mix male and female actors.

We also examined the movie release year similarity. Each release year was represented with a different binary feature. Both methods tended to give another release years as most similar metadata, however, CMF did slightly better. We observed similar results for CMF, when $\mathbf{W}'''$ was computed after a regular MF-run.

To sum up, the K-dimensional feature vectors of CMF yields a more intuitive similarity than that of the movie-NSVD1. We explain this remarkable difference as follows. Assuming that only $\mathbf{X}''$ or $\mathbf{W}'''$ are to be optimized, only the following equations should be taken into consideration:

- NSVD1: $\mathbf{Q} = \mathbf{X}''\mathbf{W}''$
- CMF: $\mathbf{X}'' = \mathbf{Q}\mathbf{W}'''^{\mathrm{T}}$

When we substitute one equation into the other, we get $\mathbf{Q} = \mathbf{Q}\mathbf{W}'''^{\mathrm{T}}\mathbf{W}''$, or $\mathbf{X}'' = \mathbf{X}''\mathbf{W}''\mathbf{W}'''^{\mathrm{T}}$. In other words: $\mathbf{W}'''^{\mathrm{T}}$ and $\mathbf{W}''$ are a kind of pseudoinverses of each other. Another interpretation: $\mathbf{W}''$ is optimized to get movie feature vectors from movie descriptions, while $\mathbf{W}'''^{\mathrm{T}}$ is optimized to get movie descriptions from movie feature vectors, i.e. the direction of the mapping is reverse.

8. CONCLUSIONS, FUTURE WORK

We presented a simple approach for CBF, and particularly for CBF with movie-metadata. The method is a simple modification of Paterek's NSVD1 [6], termed user-NSVD1, where users are represented with sparse binary vectors indicating which movies are rated by the user or not. In our proposed movie-NSVD1, the role of users and movies are interchanged, and movies are represented with the sparse vector representation of their metadata. We showed how the presented approach can be extended to handle both movie and user metadata, or implicit feedback datasets.

We investigated how effective CBF methods are in predicting ratings on new movies. We showed that even 10 ratings of a new movie are more valuable than the best solely metadata-based representation. We think that this is due to the large gap between the movie descriptions and the movies themselves: people rate movies, not their descriptions. We reckon that CBF methods can be applied more successfully to news recommendation, where the texts themselves are to be recommended and the amount of new items is higher.

We also investigated the capability of the algorithm to find similar movie actors. We concluded, that in this concern Collective Matrix Factorizations, proposed by Singh et al. [7] gives better results than movie-NSVD1.

In future work, we intend to investigate when ratings overweight the rating + movie metadata combination in prediction. In addition to that we also intend to explore how user metadata can contribute to the prediction. We conjecture that a mere few ratings of a user is more valuable than the gender, age, zip-code, etc. attributes.

Note that the proposed approach for movie-CBF decomposes $\mathbf{R}$ into $\mathbf{R} = \mathbf{P}\mathbf{Q}^{\mathrm{T}} = \mathbf{P}(\mathbf{W}''^{\mathrm{T}}\mathbf{X}''^{\mathrm{T}}) = (\mathbf{P}\mathbf{W}''^{\mathrm{T}})\mathbf{X}''^{\mathrm{T}}$. We would like to investigate $\mathbf{P}\mathbf{W}''^{\mathrm{T}}$, since this matrix contains a C''-dimensional feature vector for each user, which can describe users preferences on actors, genres, etc.

9. REFERENCES

[1] J. Basilico and T. Hofmann. Unifying collaborative and content-based filtering. In *ICML-04: Proc. of the 21st Int. Conf. on Machine learning*, page 9, New York, NY, USA, 2004.

[2] 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, 2007.

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

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

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

[6] 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.

[7] A. P. Singh and G. J. Gordon. Relational learning via collective matrix factorization. In *KDD-08: Proc. of the 14th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2008.

[8] 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.

[9] 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 2008.

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

[11] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Investigation of various matrix factorization methods for large recommender systems. In *Proc. of the 2nd Netflix-KDD Workshop*, Las Vegas, NV, USA, August 24, 2008.

[12] Y. Zhang and J. Koren. Efficient Bayesian hierarchical user modeling for recommendation system. In *SIGIR-07: Proc. of the 30th Annual Int. ACM SIGIR Conference on R&D in Information Retrieval*, pages 47–54, New York, NY, USA, 2007.