On the Gravity Recommendation System

Gábor Takács
Dept. of Measurement and Information Systems
Budapest University of Technology and Economics
Magyar Tudósok krt. 2.
Budapest, Hungary
gtakacs@mit.bme.hu

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

Bottyán Németh
Dept. of Telecom. and Media Informatics
Budapest University of Technology and Economics
Magyar Tudósok krt. 2.
Budapest, Hungary
bottyan@tmit.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 is a collaborative filtering problem. This subfield of machine learning has become popular from the late 1990s with the spread of online services that use recommendation systems, such as e.g. Amazon, Yahoo! Music, and of course Netflix. The aim of such a system is to predict what items a user might like based on his/her and other users previous ratings. The dataset of Netflix Prize is much larger than the previously known benchmark sets, therefore we first show in the paper how to store it efficiently and adaptively to various algorithms. Then we describe the outline of our solution, called the Gravity Recommendation System (GRS), to the Netflix Prize contest, which is in the leader position with RMSE 0.8808 at the time of the submission of the paper. GRS comprises of the combination of different approaches that are presented in the main part of the paper. We then compare the effectiveness of some selected individual and combined approaches against a designated subset of the dataset, and discuss their important features and drawbacks. Beside the description of successful experiments we also report on the useful lessons of (temporarily) failed ideas and algorithms.

Categories and Subject Descriptors

H.5.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Information Filtering*; G.3 [**Probability and Statistics**]: Correlation and regression analysis

General Terms

Experimentation, Algorithms

Keywords

Netflix Prize, Machine learning, Collaborative filtering, Matrix factorization

1. INTRODUCTION

Collaborative filtering (CF) is a subfield of machine learning that aims at creating algorithms to predict user preferences based on known user ratings or user behavior in selection/purchasing items. Such a system has great importance in applications in e-commerce, subscription based services, information filtering and so on. Recommendation systems providing personalized suggestions greatly increase the likelihood over a customer making a purchase than unpersonalized ones. This is especially important on such a market 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 & restaurants, gaming & humor, etc. Clearly, the online DVD rental service operated by Netflix fits into this list.

The Netflix Prize problem (see the details in the general description, [2]) belongs to one of the typical framework of CF called the *social-filtering* method. In this case the user first provides ratings of some items, titles or artifacts usually on a discrete numerical scale, and then the system recommends other items based on ratings the virtual community using the system has already provided. The virtual community was defined in [6] as "a group of people who share characteristics and interact in essence or effect only" that is in reality they do not interact. The underlying assumption of such a system that people who had similar taste in the past may also agree in the future.

1.1 Related works

The first works on the field of CF has been published from early 1990s. The GroupLens systems of [10] was one of the pioneer applications of the field where users could rate articles after reading on a 1–5 scale, and then were offered suggestion. The underlying techniques of predicting user preferences can be divided into two main groups [3]. The *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 it for prediction.

Over the last broad decade many CF algorithms have been proposed that approach the problem from different point of view, including similarity based approaches [10, 13], Bayesian networks [3], personality diagnosis [8], various matrix factorization techniques [4, 7, 12, 16], and very recently restricted Boltzman machines [11]. Short descriptions of the most important CF algorithm can be found in [1], while in [3] a more detailed survey is given on major CF approaches. The GRS described in this paper apply the combination of several different approaches to maximize its performance (see details in Section 3).

1.2 The aim of this paper

As we have experienced in the last few months, the development of a high-quality movie recommender system is not only about innovative approaches and tricky ideas but it also requires a lot of efficient coding and experimentation in order to try different variants of approaches and parameter settings, resp. Even after a superficial scanning of the related literature, one can easily find that the matrix factorization and the K-nearest neighbors approaches suits well to the Netflix Prize problem. We also believe that many other teams use these methods in their prediction algorithm. However, a good approach does not necessarily guarantee a good implementation. This latter additionally requires to work out every minor detail of the algorithm appropriately, which is laborious and cumbersome.

In this paper, we present the main components of our solution termed Gravity Recommendation Systems to the Netflix Prize and show how the individual methods are combined to obtain more efficient results. However, due to the limits of the paper and our obvious interests, we intentionally do not publish all parts of the puzzle: some small but important details remain hidden.

1.3 Organization

This paper is organized as follows. Section 2 deals with the data storage issue after having introduced the notations and some formulae used throughout the paper. Section 3 describes the algorithms we have experimented with and applied in GRS. Section 4 presents the result achieved by means of the presented approaches, which is followed by a concise discussion.

2. PRELIMINARIES

2.1 Notation

We use the following notation in this paper. The rating matrix is denoted by $\mathbf{X} \in \{1, \ldots, 5\}^{I \times J}$, where an element x_{ij} stores the rating of the jth movie provided by ith customer. I and J denote the total number of users and movies, resp. We refer to the set of all known (i, j) pairs in $\mathbf{X}$ as $\mathcal{R}$. At the prediction of a given rating we refer to the user as *active user*, and to the movie as *active movie*. Superscript ,,hat" denotes the prediction of the given quantity, that is $\hat{x}$ is the prediction of x.

Concerning the details of the Netflix Prize problem (e.g. the terminology of various datasets), we refer the Reader to the general description given in [2].

2.2 Data storage

Netflix has generously provided a 100 million ratings dataset for the contestants. Because the size of the training set is huge compared to other benchmark problems of collaborative filtering and in the usual machine learning tasks, the storage of training data is an important issue. For comparison the widespreadly used EachMovie dataset[1] only consists of about 2.8 million ratings of 73k users and 1628 movies. Another dataset of the GroupLens project (see e.g. [1]) has about 13 million ratings from 105k users on 9k movies.

Efficient data storage is particularly important since the original format of the database is impractical: reading and parsing the text files takes about 30 minutes with by means a naive implementation. It is crucial to reduce the experimenting time in order to speed up the development. The weakness of model-based approaches, namely that the model-building process is time-consuming [1], can also be eliminated or degraded with proper implementation. The time requirement of both the input processing and the learning/predicting phases can be reduced with several orders of magnitude by means of an appropriate representation of data described next.

Clearly, it is worth storing the entire training set in the memory, if it is possible. Consequently, we decided to transform the original input data into a concise binary format. Based on some experiments, we found that the date of the ratings can be thrown away. Because the 99 percent of ratings in the movie-user matrix is unknown, it is practical to use sparse representation. There are two reasonable choices:

- The *per-user storage* represents each row as a list of (column index, value) pairs.
- The *per-movie storage* represents each column as a list of (row index, value) pairs.

In the first case it is easy to go through the ratings of a given customer. In the second case the ratings of a movie can be queried efficiently. Most algorithms prefer the first representation, some prefer the second one, and some methods need both. It is also possible to store the matrix as (row index, column index, value) triplets, but this is impractical because it provides none of the above advantages.

In the training database there are 480189 customers and 17770 movies, so the space requirement of the row index and the column index is 19 bits and 15 bits, resp. The 5 different values of the ratings can be stored in 3 bits. Therefore the space requirement of one matrix element is 18 bits in the per-user and 22 bits in the per-movie representation. In practice this means that with appropriate C structs (see Table 1) a rating can be stored in 24 bits and the storage of the whole matrix requires ∼300 MB in each case.

This database representation made us possible to develop efficient algorithms on an average PC. We achieved our first

[1]It used to be available upon request from Compaq, but in 2004 the proprietary retired the dataset, and since then it is no longer available for download.

Table 1: C structures for storing one rating

`struct PerUserRating {`
 `  ushort movieId;`
 `  uchar value;`
 `};`
	`struct PerMovieRating {`
	 `  uchar userIdxHi;`
	 `  uchar userIdxLo:4;`
	 `  uchar value:4;`
	 `};`

leading position using only a 2 GHz notebook with 1 GB RAM.

For a weaker hardware (512 MB RAM), the 300 MB representation can be still too large to work with. Therefore, we have developed an even more compact per-user representation, which requires only 200 MB at the expense of a bit slower access to the data.

To achieve this, first we split the matrix into two halves. The first one contains ratings for movies with ID $\leq 17770/2 = 8885$, the second contains the rest. We utilize the fact that $8885 \cdot 5 = 44425 \leq 2^{16} = 65536$. This means, that a movie with ID between 1 and 8885 and its rating between 1 and 5 can be stored in 2 bytes:

$$(j, x_{ij}) \to c_{ij} := 5 \cdot j + x_{ij} - 1$$
$$c_{ij} \to j = \lfloor c_{ij}/5 \rfloor,\ x_{ij} = 1 + (c_{ij} \bmod 5)$$

where $j \in [1, J]$ identifies the movie. Movies with ID > 8885 can be treated similarly: first we subtract 8885 from the ID, then the rating and the new ID (≤ 8885) can be stored in 2 bytes in the second matrix.

3. APPROACHES

3.1 Matrix factorization

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

$$\mathbf{X} \approx \mathbf{UM}, \tag{1}$$

where $\mathbf{U}$ is an $I \times K$ and $\mathbf{M}$ is a $K \times J$ matrix. The u_{ik} and m_{kj} values can be considered, reps. the kth feature of the ith user and the jth movie. If we consider the matrices as linear transformations, the approximation can be interpreted as follows: the $\mathbf{M}$ matrix transforms from $\mathcal{V}_1 = \mathbb{R}^J$ into $\mathcal{V}_2 = \mathbb{R}^K$, and $\mathbf{U}$ transforms from $\mathcal{V}_2$ into $\mathcal{V}_3 = \mathbb{R}^I$. Thus, the $\mathcal{V}_2$ vector space acts as a bottleneck when predicting $\mathbf{v}_3 \in \mathcal{V}_3$ from $\mathbf{v}_1 \in \mathcal{V}_1$. In other words, the number of parameters to describe $\mathbf{X}$ is reduced from $|\mathcal{R}|$ to $IK + KJ$. However, $\mathbf{X}$ contains integers of range 1 to 5, while $\mathbf{M}$ and $\mathbf{U}$ contain real numbers.

Several matrix factorization techniques have been applied successfully to CF, including SVD (singular value decomposition, [12]), pLSA (probabilistic latent semantic analysis, [7]), and MMMF (maximum margin matrix factorization, [16]). Due to the speciality of the Netflix Prize problem, the solution should be sought as a low-rank approximation with missing data (see also e.g. [5, 9, 15]). Here we present the basics of MF methods via SVD:

$$\mathbf{X} = \mathbf{U\Sigma M}, \tag{2}$$

where $\mathbf{U} \in \mathbb{R}^{I \times \min(I,J)}$ and $\mathbf{M} \in \mathbb{R}^{\min(I,J) \times J}$ are orthogonal matrices, and $\mathbf{\Sigma} \in \mathbb{R}^{\min(I,J) \times \min(I,J)}$ contains non-zero elements only along the diagonal. These elements are called singular values, and are always non-negative. If we keep only the K largest singular values and replace the others by zero, we got $\hat{\mathbf{X}}$, a K-rank approximation of $\mathbf{X}$ with the following property:

$$\|\hat{\mathbf{X}} - \mathbf{X}\| = \min\{\|\mathbf{X}' - \mathbf{X}\| : \mathbf{X}' \in \mathbb{R}^{I \times J},\ \operatorname{rank}(\mathbf{X}') \leq K\}$$

In other words: from the SVD the optimal K-rank approximation of $\mathbf{X}$ can easily be computed, which minimizes the Frobenius norm, defined as $\|\mathbf{A}\| = \sqrt{\sum_{a \in \mathbf{A}} a^2}$, of the distance of the matrices. We can eliminate $\mathbf{\Sigma}$ from the factorization given in eq. (2):

$$\mathbf{X} = (\mathbf{U\Sigma})\mathbf{M} = \mathbf{U}(\mathbf{\Sigma M}) = (\mathbf{U}\sqrt{\mathbf{\Sigma}})(\sqrt{\mathbf{\Sigma}}\mathbf{M}),$$

thus the form reduces to eq. (1).

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

$$\hat{x}_{ij} = \sum_{k=1}^{K} u_{ik} m_{kj} \tag{3}$$

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

$$\text{SE} = \sum_{(i,j)\in\mathcal{R}} e_{ij}^2 = \sum_{(i,j)\in\mathcal{R}} \left(x_{ij} - \sum_{k=1}^{K} u_{ik} m_{kj} \right)^2$$

$$(\mathbf{U}, \mathbf{M}) = \underset{(\mathbf{U},\mathbf{M})}{\arg\min}\, \text{SE} \tag{4}$$

Here $\hat{x}_{ij}$ denotes how the ith user would rate the jth movie, according to the model, e_{ij} denotes the training error on the (i,j)th example, and SE denotes the total squared training error. Eq. (4) states that the optimal $\mathbf{U}$ and $\mathbf{M}$ minimizes the sum of squared errors only on the known elements of $\mathbf{X}$.

In order to minimize SE (which is equivalent to minimize RMSE), we have applied a simple gradient descent method to find a local minimum. Suppose that a training example x_{ij} and its approximation $\hat{x}_{ij}$ are given. We compute the gradient of e_{ij}^2:

$$\frac{\partial}{\partial u_{ik}} e_{ij}^2 = -2e_{ij} \cdot m_{kj}, \quad \frac{\partial}{\partial m_{kj}} e_{ij}^2 = -2e_{ij} \cdot u_{ik}. \tag{5}$$

We update the weights in the opposite direction of gradient:

$$u'_{ik} = u_{ik} + \eta \cdot 2e_{ij} \cdot m_{kj}, \quad m'_{kj} = m_{kj} + \eta \cdot 2e_{ij} \cdot u_{ik},$$

that is, we change the weights in $\mathbf{U}$ and $\mathbf{M}$ to decrease the error, thus better approximating x_{ij}. Here η is the learning rate. To better generalize on unseen examples, we have applied regularization with factor λ to prevent large weights:

$$u'_{ik} = u_{ik} + \eta \cdot (2e_{ij} \cdot m_{kj} - \lambda \cdot u_{ik}) \tag{6}$$

$$m'_{kj} = m_{kj} + \eta \cdot (2e_{ij} \cdot u_{ik} - \lambda \cdot m_{kj}) \tag{7}$$

Our algorithm can be summarized as follows:

1. Initialize the weights in $\mathbf{U}$ and $\mathbf{M}$ randomly. Set η and λ to some small positive value.
2. Loop until the terminal condition is met
   (a) Iterate over each known element of $\mathbf{X}$ which is not in the probe subset. For x_{ij}:
      i. compute e_{ij};

ii. compute the gradient of e_{ij}^2;

iii. update the ith row of $\mathbf{U}$ and the jth column of $\mathbf{M}$ according to eqs. (6)–(7).

(b) Calculate the RMSE on the probe subset.

The loop is terminated when the RMSE does not decrease during two iterations.

Note that if the rows of $\mathbf{U}$ and the columns of $\mathbf{M}$ are constant vectors, each row of $\mathbf{U}$ and each column of $\mathbf{M}$ remains a constant vector. That is, why we initialize randomly. Our approach is very similar to Simon Funk's SVD,[2] but we update each factor simultaneously, and initialize the matrix randomly. Simon Funk's approach learns the first factor for a certain number of iterations, then the second, and so on. His approach converges much slower than ours because it iterates more on $\mathbf{X}$. Additionally, it is not specified in his algorithm when one has finish the learning of one factor and start the next.

We remark that after the learning phase, each value of $\mathbf{X}$ can be computed easily by eq. (3), even the "unseen" values. In other words, the model ($\mathbf{U}$ and $\mathbf{M}$) can say something about how an arbitrary user would rate any movie.

GRS comprises of a combination of several methods. Among them there are some variants of the presented MF algorithm, which will be described next.

3.1.1 Using dates

We can easily incorporate the date of ratings into the MF model. Let g_{ij} denote the date of the (i,j)th rating, represented as an integer between 1 and L. We can then reformulate eq. (3) as

$$\hat{x}_{ij} = \sum_{k=1}^{K} u_{ik} m_{kj} d_{kl}, \quad \text{where } l = g_{ij}.$$

Here d_{kl} are the elements of $\mathbf{D} \in \mathbb{R}^{K \times L}$. Its weights are initialized randomly as well. We compute the gradient of $e_{ij}^2 = (x_{ij} - \hat{x}_{ij})^2$:

$$\frac{\partial}{\partial u_{ik}} e_{ij}^2 = -2e_{ij} \cdot m_{kj} \cdot d_{kl},$$
$$\frac{\partial}{\partial m_{kj}} e_{ij}^2 = -2e_{ij} \cdot u_{ik} \cdot d_{kl},$$
$$\frac{\partial}{\partial d_{kl}} e_{ij}^2 = -2e_{ij} \cdot u_{ik} \cdot m_{kj}.$$

The weight updating scheme is similar to eqs. (6)–(7):

$$u'_{ik} = u_{ik} + \eta \cdot (-\frac{\partial}{\partial u_{ik}} - \lambda \cdot u_{ik})$$
$$m'_{kj} = m_{kj} + \eta \cdot (-\frac{\partial}{\partial m_{kj}} - \lambda \cdot m_{kj})$$
$$d'_{kl} = d_{kl} + \eta \cdot (-\frac{\partial}{\partial d_{kl}} - \lambda \cdot d_{kl})$$

This method can adapt to some changes in the users' rating conventions, which may be caused for example by promotions for users to rate more movies or by improvement in the recommendation system. Although, these MF models with dates do not result a significant improvement of the prediction, they can be efficiently used in the combination since they differ sufficiently from regular MFs.

[2] `http://sifter.org/~simon/journal/20061211.html`

3.1.2 Constant values in matrices

Beside the (user ID, movie ID, date, rating) quadruples, Netflix has also provided the title and year of release of each movie. This information can be incorporated in the MF model as follows: we can extend $\mathbf{M}$ with rows that indicate the occurrence of words in the movie title, or the year of release. For example, the k_1th row of $\mathbf{M}$ is 1 or 0 depending on whether the term "Season" occurs in the title of the movies or not. This row of $\mathbf{M}$ is fixed to be a constant, thus we do not apply equ. (7) for $k = k_1$. If the occurrence of term "Season" in the title increases the ith user's rating (i.e. the user likes TV-series), the resulting model can have positive weight for u_{ik_1}. Other constant values can be inserted, for example, we can increase the size of matrices, K, with 2 by inserting the average rating of the user and the movie, resp.

3.1.3 Rounding the values

$\mathbf{X}$ contains only integers between 1 and 5, and this holds also for the unknown ratings in the Quiz set. It is straightforward that this information should also be exploited. We have tried several approaches to round the values in $\mathbf{UM}$, but none of them helped. The error of approximation (elements of $\mathbf{X} - \mathbf{UM}$) has Gaussian distribution, with $\mu = 0$ and σ between 0.7 and 1.0. Note that σ is the same as the RMSE in the Netflix contest, but on the training set. Because a large portion of the errors are ≥ 0.5, rounding the values has no point. To see this, let us suppose that we want to round the values with the following function:

$$\hat{x}'_{ij} = \rho \cdot \hat{x}_{ij} + (1-\rho) \cdot \text{round}(\hat{x}_{ij}) \quad 0 \leq \rho \leq 1 \qquad (8)$$

If we set ρ to 1, there is no rounding, if we set to 0, we round each value to the closest integer. Errors between 0 and 0.5 will decrease, errors between 0.5 and 1 will increase, between 1 and 1.5 decrease, etc. For example, suppose that for some i, j, $x_{ij} = 2$, $\hat{x}_{ij} = 1.2$, and $\rho = 0.9$. Hence, $\hat{x}'_{ij} = 1.18$ and we increased the error by 0.02. Let $\varepsilon = x_{ij} - \hat{x}_{ij}$ and $\varepsilon' = x_{ij} - \hat{x}'_{ij}$ denote the variate of training error before and after rounding, resp. If $-0.5 < \varepsilon < 0.5$, then $-0.5 \cdot \rho < \varepsilon' < 0.5 \cdot \rho$. If $0.5 < \varepsilon < 1.5$, then $1 - 0.5 \cdot \rho < \varepsilon' < 1 + 0.5 \cdot \rho$, etc.

Let φ denote the probability density function of ε:

$$\varphi(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Formally, the training error after rounding will be:

$$\text{RMSE} = \sqrt{\sum_{n \in \mathbb{Z}} \int_{n-0.5}^{n+0,5} \varphi(x) \cdot (n + \rho(x-n))^2 \, \mathrm{d}x}$$

When $\rho = 0.9$, this approach improves performance only if σ is below ~ 0.5, which seems unreachable (on the probe subset).

We have also tried another rounding approach, which defines a smooth rounding function as follows:

$$\text{sr}(x) = \lfloor x \rfloor + \frac{\tanh((x - \lfloor x \rfloor - 0.5) \cdot 2 \cdot A) - \tanh(-A)}{\tanh(A) - \tanh(-A)} \qquad (9)$$

Here A controls the "smoothness" of rounding.

We can apply rounding functions (8) or (9) either on the entire or on a part of the output, i.e.:

$$\hat{x}_{ij} = \sum_{k=1}^{K_1} u_{ik} m_{kj} + \text{sr}\left(\sum_{k=K_1+1}^{K} u_{ik} m_{kj}\right),$$

where $0 \le K_1 < K$ denotes the part not altered by rounding. With this modification, the computation of the gradient, eq. (5), becomes more complicated.

We expected that this model will activate rounding only on such users for who this is worthwhile, and thus improves the performance. However, it yielded inferior results.

We tried to apply different non-linear functions on different parts of the output, but none of these experiments were successful. The failure of these trials can be supported by the following explanatory example. If $\hat{x}_{ij}$ is 4.1, then the user would rate 4 with probability 0.9, and 5 with probability 0.1, and the decision of the user is taken completely arbitrarily.

3.1.4 Parameters of matrix factorization

The more parameters a matrix factorization has, the harder to set them well, but the more chance to get a better RMSE. We experimented with the following parameters:

- the number of features: K;
- different learning rate and regularization factor
  - for users and movies;
  - at different learning cycles;
  - values of η and λ depends on k in eqs. (6)–(7);
  - for users or movies with different number of ratings;
  - for the corresponding variables of constant features;
- probability distribution function to initialize $\mathbf{U}$ and $\mathbf{M}$;
- offset to subtract from $\mathbf{X}$ before learning (can be, e.g., set to the global average of ratings);
- nonlinear functions on parts of the output as in eq. (9).

We subsampled the matrix for faster evaluation of a parameter settings. We experienced that movie-subsampling substantially increased the error, in contrast to user-subsampling. It is interesting, that the larger the subsample is, the fewer iterations are required to achieve the optimal model. We think this is because of the redundancy in the data set.

3.1.5 Connections with neural networks

The MF model can be considered as the learning of a multi-layer perceptron given on Figure 1. The network has I inputs, J outputs and K hidden neurons, and identity activation function in each neuron. We consider the weight between the ith input and the kth hidden neuron as u_{ik}, and the weight between the kth hidden neuron and the jth output neuron as m_{kj}.

The network applies incremental learning method. For the (i,j)th rating, we set the input $\mathbf{x}$ as x_i is 1 and $x_{k \neq i} = 0$, thus $z_k = u_{ik}$ holds. Let $\hat{x}_{ij} = y_j$ denote the jth output of the network. We compute the error: $e_{ij} = x_{ij} - \hat{x}_{ij}$. This error is back-propagated from the jth output neuron towards the input layer.

In the evaluation phase, we set the input as in the learning phase, and y_j $(j = 1, \dots, J)$ predicts the active user's rating on the jth movie. The network can be generalized in many ways, e.g.:

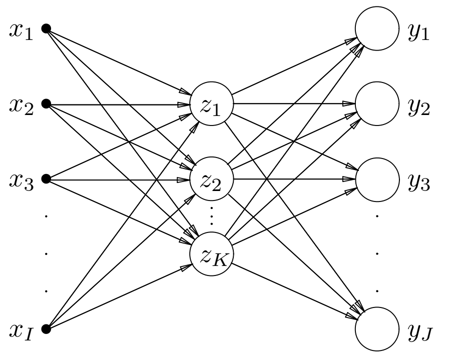

Figure 1: Multilayer perceptron for collaborative filtering

- reverse the direction of edges;
- replace some of the identity activation functions with non-linear ones in the hidden layer or in the output layer;
- create more hidden layers;
- apply batch learning.

3.1.6 2D Modifications of the matrix factorization algorithm

The linear combination of different MF algorithms is better than the best single algorithm. The result is as good as different the MFs are in the combination. Based on this observation, we made different modifications on simple MF models. In one experiment we arranged the movie and user features (recall that these are provided by MF) to a 2D lattice and defined a simple neighborhood relation between the features. If the difference of the horizontal and vertical positions of two features are small (they are neighbors) the "meaning" of those features should also be close. To achieve this, we proceed as follows. After each usual learning step, we modify the feature values towards the direction of the neighboring features by smoothing the calculated changes of feature values. In practice, in order to prevent slow training, we take only those features into account which are next to each other

$$\begin{aligned}\Delta u'_{i,k_1,k_2} &= \Delta u_{i,k_1,k_2} + \xi(\Delta u_{i,k_1+1,k_2} + \Delta u_{i,k_1-1,k_2} \\ &\quad + \Delta u_{i,k_1,k_2+1} + \Delta u_{i,k_1,k_2-1}) + \frac{\xi}{\sqrt{2}}(\Delta u_{i,k_1+1,k_2+1} \\ &\quad + \Delta u_{i,k_1-1,k_2-1} + \Delta u_{i,k_1-1,k_2+1} + \Delta u_{i,k_1+1,k_2-1}).\end{aligned}$$

Here ξ is the smoothing rate, $\Delta u'$ the modified and Δu the 2D array of the original changes in user features. The indices of Δu denotes the user index, and the two coordinates of the feature in the feature array, respectively. We applied analog formula also on the movies.

The 2D features of a movie (or user) can be nicely visualized. Meanings can be associated with the regions of the image. The labels on Figure 2 are assigned to sets of features and not to single ones. The labels has been determined based on such movies that have extreme values at the given feature.

Such feature maps are useful for detecting main differences between movies or episodes of the same movie. Figure 3

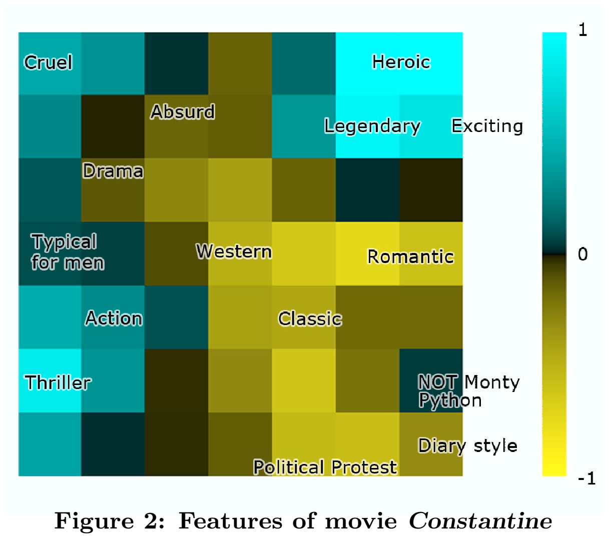

Figure 2: Features of movie *Constantine*

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 presented a unique view of the world, but the other two were rather based on the success of the first episode.

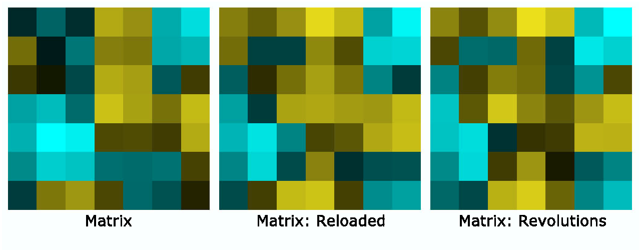

Figure 3: Features of *The Matrix* episodes

3.2 Neighbor based approaches

In neighbor based approaches a similarity value and a univariate predictor function is assigned to each movie pair or to each user pair. The first variant can be referred as the *movie neighbor* and the second as the *user neighbor* method. Assuming the movie neighbor method, the unknown matrix element x_{ij} can be estimated as

$$\hat{x}_{ij} = \frac{\sum_{k:(i,k)\in\mathcal{R}} s_{jk} f_{jk}(x_{ik})}{\sum_{k:(i,k)\in\mathcal{R}} s_{jk}}, \tag{10}$$

where s_{jk} is the similarity between the jth and the kth movie, and f_{jk} is the function that predicts the rating of the jth movie from the rating of the kth. The answer of the system is the similarity-weighted average of movie-to-movie sub-predictions.

A common approach is to use a correlation based movie-to-movie prediction and similarity. If we introduce the notation $\mathcal{R}_{jk} = \{i : (i,j),(i,k) \in \mathcal{R}\}$, then the formula of the empirical Pearson correlation [14] coefficient between two movies is the following:

$$r_{jk} = \frac{1}{|\mathcal{R}_{jk}|} \sum_{i\in\mathcal{R}_{jk}} \frac{(x_{ij}-\mu_j)(x_{ik}-\mu_k)}{\sigma_j\sigma_k},$$

where μ_j and μ_k are the empirical means, σ_j and σ_k are the empirical standard deviations of the jth and the kth movie, resp.

If there are only a few common ratings between two movies then the empirical correlation is not a good estimator of the true one. Therefore we regularize the estimate with the a priori assumption of zero correlation. This can be performed in a theoretically grounded way by applying Fisher's z-transformation:

$$z_{jk} = 0.5 \ln \frac{1+r_{jk}}{1-r_{jk}}.$$

According to [14], the distribution of z_{jk} is approximately normal with standard deviation $1/\sqrt{|\mathcal{R}_{jk}|-3}$. We can now compute a confidence interval around the z value and use the lower bound as the regularized estimate:

$$z'_{jk} = z_{jk} - \frac{\lambda}{\sqrt{|\mathcal{R}_{jk}|-3}},$$

where $\lambda \geq 0$ is the regularization factor. Our experiments showed that $\lambda = 2.3$ is a reasonable choice for the Netflix problem. The regularized correlation coefficient can be obtained from the inverse of the z-transformation:

$$r'_{jk} = \frac{\exp(2z'_{jk})-1}{\exp(2z'_{jk})+1}$$

Now we can define the similarity metric and the movie-to-movie predictor functions:

$$s_{jk} = 1/|r'_{jk}|^{\alpha},$$
$$f_{jk}(x) = \mu_j + r'_{jk}(x-\mu_k),$$

where α can be called the amplification factor. Large $\alpha \gg 1$ values increase the weight of similar movies in eq. (10).

An advantageous property of the neighbor based methods is that they can be implemented without training phase. Having said this, it is useful to precompute the correlation coefficients and keep them in the memory, because this drastically decreases testing time. It is enough to store only the correlations between the 6000 most rated movies, because these values are much more frequently needed than other ones.

Additional improvements can be achieved with the following tricks:

- Only the K most similar movies are taken into account in eq. (10).

- The average rating of the active movie is taken into account with weight β in eq. (10). Note that this is different from simply combining the output of the neighbor based predictor with the movie average, because the influence of the movie average depends on how close neighbors the active movie has.

Theoretically, we could also use the user based "dual" of this method but the user neighbor approach does not fit well to the Netflix problem, because

- user correlations are unreliable, since there are typically very few common movies between two arbitrary users;
- the distribution of the users is approximately uniform in the test set, therefore there is no such a part of the user correlation matrix that is used significantly more often than other ones. Consequently, the precomputation step is needless, and the testing time of the algorithm is significantly larger than that of the movie neighbor based approach.

We have experienced also with modified version of the user neighbor method, when we computed the correlation between each user and the top 10000 users and stored 1000 neighbors, but it did not yield any improvement on the efficiency of GRS.

3.3 Clustering based approaches

Another possible way to tackle the Netflix Prize problem is to use a clustering based approach. We independently apply clustering on users and movies that yields L user clusters and M movie clusters. Let us call the $L \times M$ (user cluster, movie cluster) pairs product clusters. The unknown matrix element x_{ij} can be predicted based on the product cluster corresponding to the ith customer and the jth movie. In the simplest case the prediction is the average rating in the product cluster.

The outline of training algorithm is the following:

1. Randomly create the initial clustering of users and movies, resp.
2. For each user u: reassign u to a new user cluster, so that the training RMSE is minimized.
3. For each movie m: reassign m to the movie cluster, so that the training RMSE is minimized.
4. If the training RMSE decreased significantly, then go to step 2.

The most frequent operations in the algorithm is computing the training RMSE and updating the attributes of the source and target clusters after the reassignment step. Both can be performed efficiently if we store the sum of ratings, the squared sum of ratings and the number of elements in each cluster. We can obtain interesting variants of this approach, if each user (movie) belongs to a separate cluster and only the movie (user) clustering is optimized.

4. EVALUATION

The approaches were evaluated on the probe subset of the training set [2]. Tables 2–4 presents some results of MF methods, neighbor methods and clustering based methods, respectively and Table 5 tabulates the best results of single methods and their combination. These results were obtained without using some efficient tricks implemented in GRS — we now hold back those ones for obvious reason. However, the results themselves indicate the efficiency of various techniques, and we also report on the effect of certain previously mentioned modifications.

As shown also in Table 5, MF is the most effective one from the three approaches. The appropriate selection of its parameters can boost MF's performance significantly (see also Table 2). In our MF experiments we initialized the weights of U and M uniformly between -0.01 and 0.01. The effect of random initialization based on 50 with different random seeds have also been examined. We found standard deviation of $1.5 \cdot 10^{-4}$ on training RMSE, $0.9 \cdot 10^{-4}$ on probe RMSE. The correlation was 0.29 between them. We realized that increasing K yields better RMSE, however, the memory consumption increases linearly, and the training time almost linearly. The increase of λ, and analogously, the decrease of η have the same effect[3]: improve RMSE but slow down the training. In case of $\eta = 0.005, \lambda = 0.02$ 37 iterations were required, as opposed to $\eta = 0.02$, where 12 iterations were enough. We can state in general that MF with better parameters requires more time to converge, but it also depends on the details of implementation. We have witnessed a 0.0010 . . . 0.0030 difference between RMSE on probe and quiz subsets. Figures 4–6 illustrate the over-learning effect and some learning curves of MF.

Table 2: RMSE of the basic matrix factorization algorithm for various η and λ values ($K = 40$)

η \ λ	0.005	0.007	0.010	0.015	0.020
0.005	0.9280	0.9260	0.9237	0.9208	0.9190
0.007	0.9326	0.9301	0.9273	0.9243	0.9226
0.010	0.9397	0.9367	0.9337	0.9306	0.9287
0.015	0.9543	0.9518	0.9494	0.9473	0.9458
0.020	0.9781	0.9767	0.9753	0.9736	0.9719

Table 3: RMSE of various neighbor based methods

Parameters				RMSE
$K = \infty$,	$\alpha = 2$,	$\beta = 0.0$,	$\lambda = 0.0$	0.9483
$K = 16$,	$\alpha = 2$,	$\beta = 0.0$,	$\lambda = 0.0$	0.9399
$K = \infty$,	$\alpha = 2$,	$\beta = 0.2$,	$\lambda = 0.0$	0.9451
$K = \infty$,	$\alpha = 2$,	$\beta = 0.0$,	$\lambda = 2.3$	0.9445
$K = 16$,	$\alpha = 2$,	$\beta = 0.2$,	$\lambda = 2.3$	0.9313

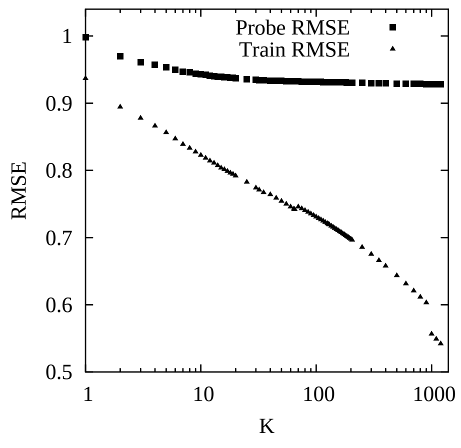

Figure 4: MFs with different values of K ($\eta = 0.01, \lambda = 0.01$)

The results of Table 3 achieved with the correlation coefficients computed only between the 6.000 most rated movies based on the top 75.000 users. When predicting other rat-

[3]though η has larger influence on RMSE than λ

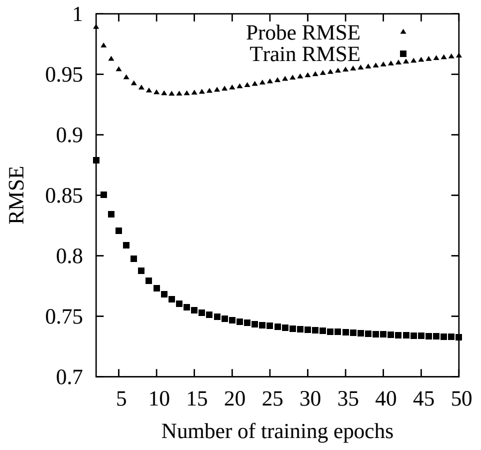

Figure 5: Learning curves of an MF run ($\eta = 0.01, \lambda = 0.01, K = 40$)

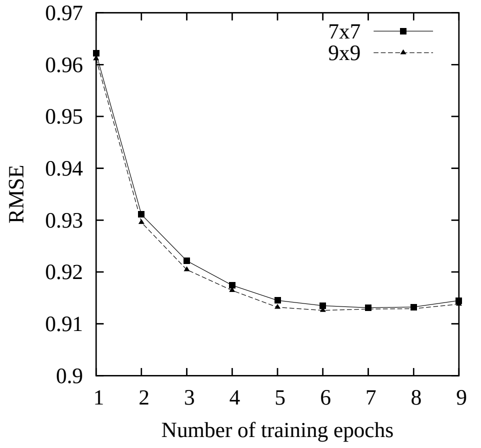

Figure 6: The learning curves of two 2D MF algorithm for various numbers of features

Table 4: RMSE of various clustering based methods

Parameters		RMSE
$L = 480189$,	$M = 5$	0.9659
$L = 5$,	$M = 17770$	0.9640
$L = 40$,	$M = 40$	0.9606

ings we simply answered a combination of the user and the movie mean.

We applied different rating normalization methods in the three cases presented in Table 4. In the first case, we subtracted the movie mean, in the second case the user mean, and in the third case the average of the movie mean and the user mean from the ratings before training. The results of the clustering based approach are inferior to Cinematch's. However, its model is easy to interpret and can be used as an input of other algorithms.

Observing the combinations of the different approaches on Table 5, one can see that they outperform the single ones significantly. Note that clustering based method does not yield improvement on the combination of MF and NB.

The effect of using only a fixed number of neighbors improved RMSE by 0.0098. Taking the movie average into account yielded an improvement of 0.0024. Regularizing the correlation coefficient using Fisher's z-transformation resulted in a 0.0043 change. When all of the three tricks were applied, then the improvement was 0.0173, which means that combination of these modification amplify their own beneficial effect.

Table 5: Best results of single approaches and their combinations

Method/Combination	RMSE
MF	0.9190
NB	0.9313
CL	0.9606
NB + CL	0.9275
MF + CL	0.9137
MF + NB	0.9089
MF + NB + CL	0.9089

5. REFERENCES

[1] S. K. L. Al Mamunur Rashid, 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, PA, USA, 2006.

[2] J. Bennett and S. Lanning. The Netflix Prize. In *Proc. of KDD Cup and Workshop*, San Jose, CA, USA, 2007. (to appear).

[3] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. Technical Report MSR-TR-98-12, Microsoft Research, Redmond, USA, 1998.

[4] 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. ACM Press.

[5] N. Del Buono and T. Politi. A Continuous Technique for the Weighted Low-Rank Approximation Problem. In *Proc. of ICCSA, Int. Conf. on Computational Science and Its Applications, Part II.*, number 3044 in Lecture Notes in Computer Science Series, pages 988–997. Springer, 2004.

[6] 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 Conference on Human Factors in Computing Systems*, pages 194–201, Denver, Colorado, United States, 1995. ACM Press/Addison-Wesley Publishing Co.

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

[8] 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, CA, USA, 2000. Morgan Kaufmann.

[9] J. D. M. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In

Proc. of ICML'05: 22nd Int. Conf. on Machine Learning, pages 713–719, Bonn, Germany, 2005.

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

[11] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted Boltzmann machines for collaborative filtering. In *Proc. of ICML'07, the 24th Int. Conf. on Machine Learning*, Corvallis, OR, USA, 2007. (to appear).

[12] 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, MA, USA, 2000.

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

[14] G. W. Snedecor and W. G. Cochran. *Statistical Methods.* Iowa State University Press, 7th edition, 1980.

[15] N. Srebro and T. S. Jaakkola. Weighted low-rank approximations. pages 720–727, Washington DC, USA.

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