Major components of the Gravity Recommendation System

Gábor Takács
Széchenyi István University
Department of Mathematics
and Computer Science
Egyetem tér 1.
Győr, Hungary
gtakacs@sze.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,
Domonkos Tikk*
Dept. of Telecom. and Media
Informatics
Budapest University of
Technology and Economics
Magyar Tudósok krt. 2.
Budapest, Hungary
{bottyan,tikk}@tmit.bme.hu

ABSTRACT

The Netflix Prize is a collaborative filtering problem. This subfield of machine learning became popular in the late 1990s with the spread of online services that used recommendation systems (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 Netflix Prize dataset is much larger than former benchmark datasets, therefore the scalability of the algorithms is a must. This paper describes the major components of our blending based solution, called the Gravity Recommendation System (GRS). In the Netflix Prize contest, it attained RMSE 0.8743 as of November 2007. We now compare the effectiveness of some selected individual and combined approaches on a particular subset of the Prize dataset, and discuss their important features and drawbacks.

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 the selection/purchasing of items. Recommendation systems providing personalized suggestions greatly increase the likelihood of a customer making a purchase compared to systems providing unpersonalized ones. This is especially important in markets where the variety of choices is large, the taste of customers is important, and last but not least, the price of 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 details in [1]) belongs to the *social-filtering* CF framework. 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 [5]. The underlying assumption of such a system is that people who had similar tastes in the past may also agree in the future.

The first works on the field of CF have been published in the early 1990s. GroupLens systems in [7] was one of the pioneer applications in the field, where users could rate articles after having read them on a 1–5 scale, and were then offered suggestions. The underlying techniques of predicting user preferences can be divided into two main groups [2]. *Memory based* approaches operate on the entire rating database. 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 points of view, including similarity based approaches [7; 9], Bayesian networks [2], various matrix factorization techniques [3; 6; 11], and very recently restricted Boltzman machines [8].

2. APPROACHES

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*. Circumflex accents denote the prediction of given quantities, i.e. $\hat{x}$ for x. We refer the Reader to [1] for the description of Netflix Prize datasets.

2.1 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. Values u_{ik} and m_{kj} can be considered the kth feature of the ith user and the jth movie, respectively. If we consider the matrices as linear transformations, the approximation can be interpreted as follows: the $\mathbf{M}$ matrix transforms from $\mathbb{R}^J$ into $\mathbb{R}^K$, and $\mathbf{U}$ transforms from $\mathbb{R}^K$ into $\mathbb{R}^I$. Thus, the $\mathbb{R}^K$ acts as a bottleneck when predicting $\mathbb{R}^I$ from $\mathbb{R}^J$.

The number of parameters to describe $\mathbf{X}$ is reduced from $|\mathcal{R}|$ to $IK + KJ$. Note that $\mathbf{X}$ contains integers in [1, 5], while the elements of $\mathbf{M}$ and $\mathbf{U}$ are real numbers.

*Domonkos Tikk was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Science.

Several matrix factorization techniques being able to handle missing data have been successfully applied to CF (see e.g. [4; 6]). Note that missing data cannot be treated as zero.

For the given problem, our goal is to find $\mathbf{U}$ and $\mathbf{M}$ such that the elements of $\hat{\mathbf{X}} = \mathbf{UM}$ are as close to the elements of $\mathbf{X}$ as possible, but only for known ratings. Formally:

$$\hat{x}_{ij} = \sum_{k=1}^{K} u_{ik} m_{kj}, \quad e_{ij} = x_{ij} - \hat{x}_{ij} \quad \text{for } (i,j) \in \mathcal{R} \tag{2}$$

$$\text{SE} = \sum_{(i,j)\in\mathcal{R}} e_{ij}^2, \quad (\mathbf{U},\mathbf{M}) = \arg\min_{(\mathbf{U},\mathbf{M})} \text{SE} \tag{3}$$

Here $\hat{x}_{ij}$ denotes how the ith user would rate the jth movie, according to the model, e_{ij} the training error on the (i,j)th example, and SE the total squared training error. Eq. (3) states that optimal $\mathbf{U}$ and $\mathbf{M}$ matrices minimize the sum of squared errors only over the known elements of $\mathbf{X}$.

To minimize SE, we applied a simple gradient descent method to find a local minimum. The gradient of e_{ij}^2 is:

$$\frac{\partial}{\partial u_{ik}} e_{ij}^2 = -2e_{ij} \cdot m_{kj}, \quad \frac{\partial}{\partial m_{kj}} e_{ij}^2 = -2e_{ij} \cdot u_{ik}. \tag{4}$$

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

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

Here η is the learning rate. To better generalize on unseen examples, we applied regularization with factor λ to prevent large weights:

$$u'_{ik} = u_{ik} + \eta \cdot (2e_{ij} \cdot m_{kj} - \lambda \cdot u_{ik}) \tag{5}$$

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

Initially, we set the weights in $\mathbf{U}$ and $\mathbf{M}$ randomly, and set η and λ to some small positive value. Steps (5)–(6) were repeated until the (RM)SE improved.

Our approach is similar to Simon Funk's SVD,[1] but we update each factor simultaneously, and initialize the matrix randomly. Consequently, our approach converges much faster, since it iterates much less on $\mathbf{X}$.

We remark that after the learning phase, each value of $\mathbf{X}$ can be computed easily by (2), 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. We now describe some important MF variants that are used in GRS.

2.1.1 Constant values in matrices

Besides 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 or not "Season" occurs in the title of the movies. This row of $\mathbf{M}$ is fixed to be a constant, thus we do not apply equ. (6) for $k = k_1$. If the occurrence of "Season" in a title increases the ith user's rating (i.e. the user likes TV-series), the model will contain a positive weight for u_{ik_1}. Other constant values can also be inserted. For example, we can increase the size of matrices, K, by 2, by inserting the average rating of the user and the movie, resp. We found that the most effective approach was to insert constant 1s into the matrix (increasing K by 2 again).

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

2.1.2 Parameters of matrix factorization

The more parameters a matrix factorization has, the harder it is to set them well, but the greater is the chance of getting a more precise or different MF. Our goal is to find many different and precise MFs. We experimented mainly with the following parameters:

- the number of features: K;
- different learning rate and regularization factors for users and movies;
- probability distribution function for initialization;
- offset to subtract from $\mathbf{X}$ before learning;
- nonlinear functions on parts of the output.

We subsampled the matrix for faster evaluation of parameter settings. We experienced that movie-subsampling substantially increased the error, in contrast to user-subsampling. It is interesting that the larger the subsample was, the fewer iterations were required to achieve the optimal model.

2.1.3 2D modifications of the matrix factorization algorithm

In the spirit of creating many different MFs we experimented with the following: We arranged the movie and user features (recall that these are provided by MF) to form 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:

$$\Delta u'_{i,k_1,k_2} = \Delta u_{i,k_1,k_2} + \xi \cdot \left(\sum_{(x,y)\in S} \frac{\Delta u_{i,k_1+x,k_2+y}}{\sqrt{x^2+y^2}} \right).$$

Here S is $\{-1,0,1\}^2 \setminus \{(0,0)\}$, 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 denote the user index, and the two coordinates of the feature in the feature array, resp. We applied an analogous formula on the movies as well.

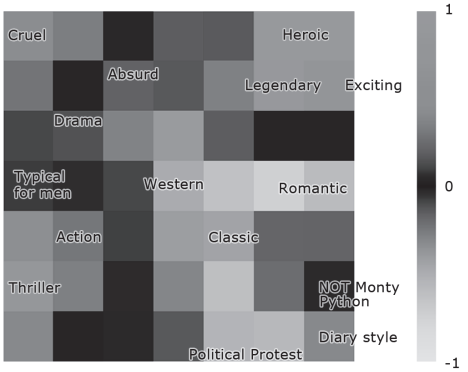

Figure 1: Features of movie *Constantine*

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 1 are assigned to sets of features and not to single ones. The labels have been determined based on movies having extreme values at the given feature.

Such feature maps are useful to detect main differences between movies or episodes of the same movie. Figure 2 represents the three episodes of The Matrix movie. One can observe that the main characteristic of the feature maps are the same, but there are noticeable changes between the first and the other episodes. In the first episode the feature values are higher in the area of "Political protest" and "Absurd" and lower around "Legendary".

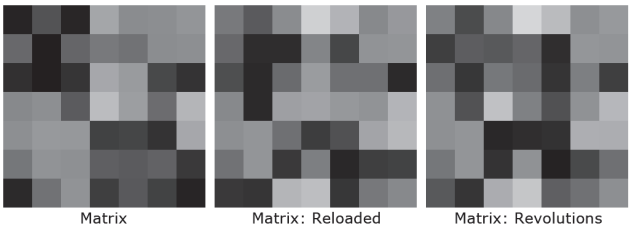

Figure 2: Features of *The Matrix* episodes

2.2 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 to 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

$$\hat{x}_{ij} = \frac{\sum_{k:(i,k)\in\mathcal{R}} s_{jk} f_{jk}(x_{ik})}{\sum_{k:(i,k)\in\mathcal{R}} s_{jk}}, \qquad (7)$$

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 based on 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 coefficient [10] 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: $z_{jk} = \tanh^{-1}(r_{jk})$. The distribution of z_{jk} is approximately normal [10] 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: $z'_{jk} = z_{jk} - \lambda/\sqrt{|\mathcal{R}_{jk}|-3}$, 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: $r'_{jk} = \tanh z'_{jk}$.

We can now define the similarity metric and the movie-to-movie predictor functions:

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

where α can be called the amplification factor. Large $\alpha \gg 1$ values increase the weight of similar movies in eq. (7).

An advantageous property of neighbor based methods is that they can be implemented without training phases. 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 sufficient 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 using the following modifications:

- We take only the K most similar movies into account.
- The average rating of the active movie is taken into account with weight β in eq. (7). 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 the proximity of the active movie to its neighbors.

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 few common movies between two arbitrary users;
- the distribution of the users is approximately uniform in the test set, therefore there is not a part of the user correlation matrix that is used significantly more often than other ones. Consequently, the precomputation step is useless, and the testing time of the algorithm is significantly greater than that of the movie neighbor based approach.

3. EVALUATION

The approaches were evaluated on the probe subset of the training set in [1]. Tables 1 and 2 present some results of MF and neighbor based methods, respectively, and Table 3 tabulates the best results of single methods and their combinations. Due to the lack of space, we present here results that were obtained without the use of any performance boosting technique. However, the results themselves indicate the efficiency of various techniques, and we also report on the effect of certain previously mentioned modifications.

Table 1: 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 2: 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

Table 3: Best results of single and combined approaches

Method/Combination	RMSE
MF	0.9190
NB	0.9313
MF + NB	0.9089

As shown also in Table 3, MF is the most effective approach of the three. The appropriate selection of its parameters can boost MF's performance significantly (see also Table 1). In our MF experiments we initialized the weights of U and M uniformly between -0.01 and 0.01. 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: it improves RMSE but slows 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. The running times was ranged from 15 mins to 2 hours depending on the parameters. We can state in general that MF with better parameters requires more time to converge, but it also depends on the details of implementation. Figure 3 illustrates the effect of K (when $\eta = 0.01, \lambda = 0.01$) and some learning curves of 2D MFs.

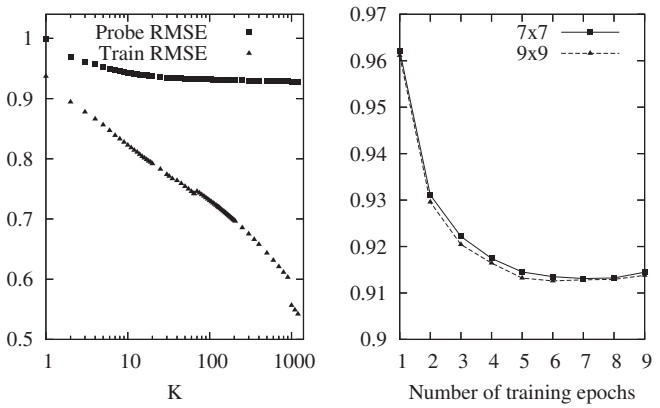

Figure 3: MFs with different values of K (on the left), learning curves of two 2D MF algorithm (on the right)

The results of Table 2 were achieved with the correlation coefficients computed only between the 6.000 most rated movies based on the top 75.000 users. When predicting other ratings we simply answered a combination of the user and the movie mean.

Observing the combinations of the different approaches on Table 3, one can see that they outperform the single ones significantly.

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 enhancements were applied, the improvement was 0.0173, which means that combination of these modifications amplify their own beneficial effects.

4. REFERENCES

[1] J. Bennett and S. Lanning. The Netflix Prize. In *Proc. of KDD Cup and Workshop*, pages 3–6, San Jose, CA, USA, 2007.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

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

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