A Modified Fuzzy C-Means Algorithm For Collaborative Filtering

Jinlong Wu
LMAM and School of Mathematical Sciences, Peking University
Beijing 100871, China
jinlon.wu@gmail.com

Tiejun Li
LMAM and School of Mathematical Sciences, Peking University
Beijing 100871, China
tieli@pku.edu.cn

ABSTRACT

Two major challenges for collaborative filtering problems are scalability and sparseness. Some powerful approaches have been developed to resolve these challenges. Two of them are Matrix Factorization (MF) and Fuzzy C-means (FCM). In this paper we combine the ideas of MF and FCM, and propose a new clustering model — Modified Fuzzy C-means (MFCM). MFCM has better interpretability than MF, and better accuracy than FCM. MFCM also supplies a new perspective on MF models. Two new algorithms are developed to solve this new model. They are applied to the Netflix Prize data set and acquire comparable accuracy with that of MF.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning;
H.3.3 [**Information storage and retrieval**]: Information search and retrieval—*Information filtering*

General Terms

Algorithms, Experimentation, Performance

Keywords

Collaborative Filtering, Clustering, Matrix Factorization, Fuzzy C-means, Netflix Prize

1. INTRODUCTION

Recommendation systems are usually constructed on the basis of two types of different methods — *content-based filtering (CBF)* and *collaborative filtering (CF)*. Content-based filtering methods provide recommendations based on features of users or items. However, it is difficult to extract features from users or items in some circumstances. For example, how can one extract features from a shirt to depict whether it is beautiful or not? Collaborative filtering methods circumvent this difficulty. They just use the known ratings of items made by users to predict ratings of new user-item pairs. The philosophy of collaborative filtering is that two users probably continue choosing similar products if they have already chosen similar ones. We will consider the collaborative filtering algorithms in this paper.

Many algorithms for CF problems have been developed, such as regressions, clusterings, matrix factorizations, latent class models and Bayesian models etc [3, 5]. In this paper we propose an efficient clustering model — Modified FCM (MFCM), which is motivated by Fuzzy C-means (FCM) but aims to minimize Root Mean Squared Error (RMSE). MFCM supplies a new perspective on matrix factorization (MF) methods. It gives a more reasonable explanation why MF works well for CF problems. Furthermore, two new algorithms MFCM1 and MFCM2 are proposed to realize MFCM in this paper.

This paper is arranged as follows. In Section 2 we briefly review FCM and MF algorithms. Our new model MFCM and algorithms MFCM1 and MFCM2 are described in Section 3. In Section 4 we show the results of these algorithms applied to the Netflix Prize data set. Finally we make conclusions in Section 5.

2. FCM AND MF ALGORITHMS

2.1 Fuzzy C-means (FCM)

The idea of clustering users is quite natural since a collaborative filtering algorithm usually tries to make recommendations to a user based on the histories of other users who showed similar preferences or tastes with this user. We can cluster the users into different classes. The users in the same class will be assumed to have similar preferences and those in different classes will be assumed to have distinct preferences.

One of the simplest clustering algorithms is *K-means*. K-means is understandable and implementable easily. However, every user is only put into one class eventually, which is too rigorous for most real-world problems. For CF problems it usually sounds more reasonable to allow that users belong to different classes. FCM takes this idea and classifies every user into different classes with suitable probabilities.

Denote the rating of movie m made by user u as $r_{u,m}$. All ratings made by the user u form a vector $\boldsymbol{r}_u$. Denote the set of all user-item pairs in the training set by $\mathcal{P}$. That is, $\mathcal{P} = \{(u,m) | r_{u,m}$ is in the training set$\}$. Denote $\{m|(u,m) \in \mathcal{P}\}$ by $\mathcal{P}_u$ and $\{u|(u,m) \in \mathcal{P}\}$ by $\mathcal{P}^m$. Let $z_{u,k}$ be the probability that user u belongs to cluster k, and $\boldsymbol{Z} = (z_{u,k})$ is the $U \times K$ probability matrix, where U and K is the

2nd Netflix-KDD Workshop, August 24, 2008, Las Vegas, NV, USA.

number of users and classes respectively. Let $c_{k,m}$ be the center vaule of movie m in class k, and $\boldsymbol{C} = (c_{k,m})$ is the $K \times M$ center matrix, where M is the number of movies. The goal of FCM is to choose the matrix $\boldsymbol{Z}$ and the center matrix $\boldsymbol{C}$ in order to minimize the objective function:

$$\begin{aligned} F(\boldsymbol{Z},\boldsymbol{C}) &= \sum_{u=1}^{U}\sum_{k=1}^{K} z_{u,k}^{\alpha} \parallel \boldsymbol{r}_u - \boldsymbol{c}_k \parallel^2 \\ &= \sum_{u=1}^{U}\sum_{k=1}^{K} z_{u,k}^{\alpha} \sum_{m\in\mathcal{P}_u} (r_{u,m} - c_{k,m})^2 \end{aligned} \tag{1}$$

with the constraints $\mathbf{Z1} = \mathbf{1}$ and $\mathbf{Z} \geq 0$ [1] since $\boldsymbol{Z}$ is a probability matrix, where α is a user-defined positive real number, $\boldsymbol{c}_k$ is the center vector of cluster k, that is, $\boldsymbol{c}_k = (c_{k,1}, \ldots, c_{k,M})$. Typically α is taken to be 2.

A standard iteration algorithm to learn $\boldsymbol{Z}$ and $\boldsymbol{C}$ has been proposed. After the algorithm converges, we achieve the final probability and center matrix $\hat{\boldsymbol{Z}}$ and $\hat{\boldsymbol{C}}$. A new pair of user-movie can be predicted by

$$\hat{r}_{u,m} = \sum_{k=1}^{K} \hat{z}_{u,k}\hat{c}_{k,m}. \tag{2}$$

2.2 Matrix Factorization (MF)

The philosophy of matrix factorization (MF) is from SVD. It aims to find two matrices $\boldsymbol{W}$ and $\boldsymbol{V}$ to minimize some norm of the residual

$$\parallel \boldsymbol{R} - \boldsymbol{W}\boldsymbol{V}^T \parallel, \tag{3}$$

where $\boldsymbol{R} = (r_{u,m})$ is the rating matrix with size $U \times M$, $\boldsymbol{W} = (w_{u,k})$ and $\boldsymbol{V} = (v_{m,k})$ are $U \times K$ and $M \times K$ respectively, both of which will be learned from the data. Usually the norm $\parallel \cdot \parallel$ is taken as the Frobenius norm and only $r_{u,m} \in \mathcal{P}$ are considered. K is a relatively small user-defined positive integer.

To prevent overfitting, some shrinkage method should be applied to shrink the parameters in $\boldsymbol{W}$ and $\boldsymbol{V}$. Usually Ridge shrinkage is useful [7, 8, 9]. To summarize, the objective function we should minimize is

$$\begin{aligned} G(\boldsymbol{W},\boldsymbol{V}) =& \frac{1}{2}\sum_{(u,m)\in\mathcal{P}} \big[(r_{u,m} - \boldsymbol{w}_u\boldsymbol{v}_m^T)^2 \\ &+ \lambda(\parallel \boldsymbol{w}_u \parallel_2^2 + \parallel \boldsymbol{v}_m \parallel_2^2)\big], \end{aligned} \tag{4}$$

where $\boldsymbol{w}_u$ and $\boldsymbol{v}_m$ is the u-th row of $\boldsymbol{W}$ and the m-th row of $\boldsymbol{V}$ respectively, and λ is the *shrinkage coefficient*. The steepest descent method is usually used to solve (4).

One commonly used explanation why MF works well for CF problems is that each row of $\boldsymbol{W}$ represents one user's preference factors and each row of $\boldsymbol{V}$ one movie's attribute factors. When these two match for a particular user and a movie, the rating is probably high.

3. A NEW CLUSTERING MODEL — THE MODIFIED FCM

In comparison with the factor-based explanation of MF, we think the idea of fuzzy clustering in FCM is more natural. But the objective function in (1) is a little confusing. Since our goal is to find $\boldsymbol{Z}$ and $\boldsymbol{C}$ to achieve the best prediction accuracy, why not minimize the prediction errors directly?

A more natural objective function is

$$\begin{aligned} H(\boldsymbol{Z},\boldsymbol{C}) =& \parallel \boldsymbol{R} - \boldsymbol{Z}\boldsymbol{C} \parallel_{\mathrm{F}}^2 \\ =& \sum_{(u,m)\in\mathcal{P}} (r_{u,m} - \sum_{k=1}^{K} z_{u,k}c_{k,m})^2 \\ =& \sum_{(u,m)\in\mathcal{P}} \Big[\sum_{k=1}^{K} z_{u,k}(r_{u,m} - c_{k,m})\Big]^2 \end{aligned} \tag{5}$$

with the constraints

$$\boldsymbol{Z1} = \mathbf{1} \quad \text{and} \quad \boldsymbol{Z} \geq 0 \tag{6}$$

since $\boldsymbol{Z}$ is a probability matrix. Hence the new constrained optimization problem that we need to solve is

$$\min_{\boldsymbol{Z1}=\mathbf{1},\boldsymbol{Z}\geq 0} H(\boldsymbol{Z},\boldsymbol{C}). \tag{7}$$

After $\boldsymbol{Z}$ and $\boldsymbol{C}$ are obtained, (2) can be used to predict any new user-movie pair. Since the new model is motivated by FCM, we refer to it as *Modified FCM (MFCM)*.

Note that if we take $\alpha = 2$ in (1), Equation (1) can be rewritten as

$$\sum_{(u,m)\in\mathcal{P}}\sum_{k=1}^{K} [z_{u,k}(r_{u,m} - c_{k,m})]^2,$$

which is similar with our new objective function (5). However, the optimization problem (7) in MFCM is much more difficult to be solved than the original one in FCM. For Equation (1) we can iteratively update the probability $z_{u,k}$ and the center $c_{k,m}$ explicitly from the equations $\partial F/\partial z_{u,k} = 0$ and $\partial F/\partial c_{k,m} = 0$ until the algorithm converges. But $z_{u,k}$ and $c_{k,m}$ can not be obtained explicitly from the above equations. Hence the same method is not applicable for our new problem. This may be the reason why FCM choose to minimize (1) instead of (5).

If the constraints in (7) are neglected, our new objective function (5) is completely the same as equation (3) in MF. Our new fuzzy clustering idea also supplies a new explanation why MF is reasonable for CF problems.

MF can be solved efficiently by steepest descent (or called gradient descent with the momentum 0) method. Since our new problem (7) is similar with that in MF, we expect that a similar algorithm can be applied for (7) .

The simplest method to handle the constraints is to penalize the parameters $z_{u,k}$ when they do not satisfy the constraints. All our algorithms are applied to the residuals of the original ratings, thus it is reasonable to shrink the center $c_{k,m}$ when it is far from 0. We also penalize the probability $z_{u,k}$ if it is far from 0 or disobeys the probability constraints (6). To summarize, the previous constrained problem is transformed into an unconstrained problem:

$$\begin{aligned} H_1(\boldsymbol{Z},\boldsymbol{C}) =& \frac{1}{2}\sum_{(u,m)\in\mathcal{P}} \Big[(r_{u,m} - \boldsymbol{z}_u\boldsymbol{c}_m)^2 + \lambda \parallel \boldsymbol{c}_m \parallel_2^2 \\ &+ \lambda(\parallel \boldsymbol{z}_u \parallel_2^2 + (\boldsymbol{z}_u\mathbf{1} - 1)^2 + \parallel \boldsymbol{z}_{u-} \parallel_2^2)\Big], \end{aligned} \tag{8}$$

where $\boldsymbol{v}_- = (v_{1-}, \ldots, v_{n-})$ and $v_{i-} = \max\{0, -v_i\}$. In our experiments the penalization parameter λ is taken to be small values. Thus the aim of the penalization terms is just to shrink the parameters to alleviate overfitting rather than to constrain the parameters to satisfy (6) strictly. (8) is actually a modified version of (4). Hence the method to minimize

[1] $\mathbf{Z} \geq 0$ means that every element of $\boldsymbol{Z}$ is not less than 0 .

(4) can be used directly to minimize (8). We refer to this algorithm as *MFCM1*.

The accuracy of MFCM1 is usually a little better than that of MF according to our experiments (see more details in Section 4), but the resulting probability matrix $\boldsymbol{Z}$ can not satisfy the constraints in (6) strictly, which loses its interpretability somewhat.

The difficulty of solving (7) originates from its constraints (6). The other natural idea to handle (6) is to enforce them into the objective function:

$$\tilde{H}(\boldsymbol{Z},\boldsymbol{C}) = \sum_{(u,m)\in\mathcal{P}} (r_{u,m} - \sum_{k=1}^{K} p_{u,k}c_{k,m})^2, \qquad (9)$$

where $p_{u,k} = e^{z_{u,k}}/\sum_{l=1}^{K} e^{z_{u,l}}$ is the probability that user u belongs to cluster k . Then $\boldsymbol{P} = (p_{u,k})$ satisfies all the constraints in (6) automatically.

With the same reason as in MFCM1, center $c_{k,m}$ should be penalized if it is far from 0. $z_{u,k}$ is also regularized towards 0 since $z_{u,k} = 0$ $(k = 1, \ldots, K)$ means that user u belongs to every cluster with the same probability. When this is taken into consideration, our final objective function becomes:

$$\begin{aligned} H_2(\boldsymbol{Z},\boldsymbol{C}) =& \frac{1}{2}\sum_{(u,m)\in\mathcal{P}} \left(r_{u,m} - \frac{1}{\sum_{k=1}^{K} e^{z_{u,k}}} \sum_{k=1}^{K} e^{z_{u,k}} c_{k,m} \right)^2 \\ & + \lambda(\| \boldsymbol{c}_m \|_2^2 + \| \boldsymbol{z}_u \|_2^2). \end{aligned} \qquad (10)$$

(10) can be solved efficiently by gradient descent with nonzero momentum. We refer to this algorithm as *MFCM2*.

4. EXPERIMENTS

4.1 The Netflix Prize Data Set

The Netflix Prize was founded by an online movie rental company Netflix at October, 2006. Its aim is to improve the accuracy of Netflix's movie recommendation system — *Cinematch*^{SM} by 10% percent. Three data sets are public for competitors: the training set, probe set (a small part of the training set) and quiz set (or qualifying set). They involves 480,189 different users who own unique user IDs ranging from 1 to 2,649,429, and 17,770 different movies with unique movie IDs ranging from 1 to 17,770. Each rating has a value belonging to $\{1, 2, \ldots, 5\}$. The whole training set is composed of 100,480,507 user-movie pairs, The probe set is composed of 1,408,395 pairs which are included in the training set, and the quiz set consists of 2,817,131 pairs. All ratings in the training set are given to learn models, and ratings in the quiz set are kept by Netflix in order to check the accuracy of competitors' models. *Root Mean Squared Error (RMSE)* is used to decide which predictions are the best. The RMSE of Cinematch^{SM} for the quiz set is 0.9514, and anybody who achieves 10% improvement of RMSE, namely 0.8514, will get 1 million dollars from the Netflix. The readers may be refered to [2] for more details.

4.2 Data Preprocessing

Suppose $\tilde{r}_u$ and $\tilde{r}^m$ are the average ratings of user u and movie m respectively, and $\bar{r}$ is the global average rating. All the averages are computed only by ratings in the training data $\mathcal{P}$.

Table 1: RMSE for different models. We take $K = 40$, $\eta = 0.004$ and $\epsilon = 10^{-5}$ in all the three models. The shrinkage coefficient $\lambda = 0.025$ in MF and MFCM1, and $\lambda = 0.0002$ in MFCM2. The momentum $\mu = 0.85$.

Models	NO. of Iterations	RMSE
MF	37	0.920124
MFCM1	40	0.918029
MFCM2	112	0.922317

Since typically a user rates a small proportion of movies, the value of $\tilde{r}_u$ is usually not very reliable compared to the global average $\bar{r}$. Hence it is reasonable to shrink $\tilde{r}_u$ to approach $\bar{r}$:

$$\bar{r}_u = \frac{|\mathcal{P}_u|\tilde{r}_u + \kappa_1\bar{r}}{|\mathcal{P}_u| + \kappa_1} = \bar{r} + \frac{|\mathcal{P}_u|}{|\mathcal{P}_u| + \kappa_1}(\tilde{r}_u - \bar{r}), \qquad (11)$$

where κ_1 is a positive constant value and called the *shrink factor of users*. Similar method can be used to shrink $\tilde{r}^m$:

$$\bar{r}^m = \frac{|\mathcal{P}^m|\tilde{r}^m + \kappa_2\bar{r}}{|\mathcal{P}^m| + \kappa_2} = \bar{r} + \frac{|\mathcal{P}^m|}{|\mathcal{P}^m| + \kappa_2}(\tilde{r}^m - \bar{r}), \qquad (12)$$

where κ_2 is the *shrink factor of movies*. $\bar{r}_u$ and $\bar{r}^m$ are thought to be more reliable averages compared to $\tilde{r}_u$ and $\tilde{r}^m$, and they are used in our experiments.

Intuitively a coarse prediction of $r_{u,m}$ might be $\bar{r}_u + \bar{r}^m - \bar{r}$, that is,

$$\begin{aligned} \hat{r}_{u,m} &= \bar{r}_u + \bar{r}^m - \bar{r} \\ &= \frac{|\mathcal{P}_u|}{|\mathcal{P}_u| + \kappa_1}(\tilde{r}_u - \bar{r}) + \frac{|\mathcal{P}^m|}{|\mathcal{P}^m| + \kappa_2}(\tilde{r}^m - \bar{r}) + \bar{r}. \end{aligned} \qquad (13)$$

We call this prediction strategy the *Average Prediction (AP)*, which is also used in [6].

Compared to the preprocessing method proposed by Bell and Koren [1], this method is symmetric for users and movies. Its resulting averages do not rely on whether user or movie averages are first calculated. Moreover, the above preprocessing method generates better predictive results in our experiments.

All of our algorithms in this paper are applied to the residual ratings $r_{u,m} - (\bar{r}_u + \bar{r}^m - \bar{r})$ ($\kappa_1 = 50$ and $\kappa_2 = 100$) except with specific statement. We still use the token $r_{u,m}$ as the residual rating without confusion.

4.3 Results

In our experiments, FCM only produces RMSE of 0.9469 on the probe set of the Netflix prize data, and MF produces RMSE of 0.9201, which is much better than that of FCM. Another advantage of MF is that it converges more rapidly. Typically MF converges after several dozens of iterations and FCM converges after hundreds of iterations. Our two new algorithms MFCM1 and MFCM2 have similar RMSE with MF but better interpretability. All the results are shown in Table 1. Generally MFCM1 generates a little better results than that MF does, and MFCM2 generates a little worse results. However, results of MFCM2 have much better interpretability since they satisfy the probability constraints (6) strictly.

A smaller learning rate η usually produces a smaller RMSE for the algorithms in Table 1 [9]. This can be seen from Table 2 obviously. However, the rate of convergence halves when η halves. A common method to fix the problem is

Table 2: Results of different models when the learning rate η has different values. All the results originate from $K = 40$ and $\epsilon = 10^{-5}$, and λ is 0.025 for MFCM1 and 0.0002 for MFCM2. In addition, MFCM2 has the momentum $\mu = 0.85$. η is reduced by (14) in which $\eta^{(0)} = 0.004$ and $\epsilon_0 = 0.02$ for MFCM1, η is reduced by (15) in which $\eta^{(0)} = 0.006$ and $N = 80$ for MFCM2.

Models	η	NO.	RMSE
MFCM1	0.004	40	0.918029
	0.002	85	0.916028
	0.001	176	0.915017
	Reducing η by (14)	55	0.915165
MFCM2	0.006	81	0.923233
	0.004	112	0.922317
	0.002	199	0.921644
	Reducing η by (15)	121	0.922183

to take a large η in the beginning and decrease η gradually when iterations continue [4].

For MF and MFCM1, our experiments show the following strategy of reducing η works well:

$$\eta^{(n+1)} = \begin{cases} \eta^{(n)}/2, & \text{if } \delta_n/\eta^{(n)} \le \epsilon_0, \\ \eta^{(n)}, & \text{otherwise} , \end{cases} \tag{14}$$

where $\eta^{(n)}$ and δ_n are the value of η and the decrease of RMSE for the n-th iteration respectively, and ϵ_0 is a small positive constant. If $\eta^{(0)} = 0.004$ and $\epsilon_0 = 0.025$, the RMSE decreases to 0.915165 after 55 iterations.

Unfortunately (14) can not improve the accuracy of MFCM2. The other strategy we try is

$$\eta^{(n+1)} = \frac{\eta^{(0)}}{1 + n/N}, \tag{15}$$

where N is a user-defined positive constant. If $\eta^{(0)} = 0.006$ and $N = 80$, the RMSE of MFCM2 decreases from 0.923233 to 0.922183 after 121 iterations. The improvement is modest compared to that of MFCM1.

Another trend for MFCM2 in our experiments is that a smaller momentum generates better predictions, but causes a slower convergence rate at the same time.

As stated in Section 4.2, all algorithms in this paper are applied to residual ratings $r_{u,m} - (\bar{r}_u + \bar{r}^m - \bar{r})$. The final predictions of an algorithm depend much on the values of $\bar{r}_u$ and $\bar{r}^m$, namely the values of κ_1 and κ_2. A common method to determine κ_1 and κ_2 is to try some different values and the best pair is used at last. A more robust method is to treat $\bar{r}_u$ and $\bar{r}^m$ as variables and adjust their values adaptively as the model is being established [7]. Their updates can be achieved by steepest descent method or letting their derivatives equal 0. Both MFCM1 and MFCM2 are easily modified to merge these ideas. The RMSE of MFCM1 decreases from 0.915165 to 0.910996 and the RMSE of MFCM2 decreases from 0.922183 to 0.920141. Both of them use the same parameters as shown in Table 2.

5. CONCLUSIONS

In this paper we propose a new clustering model — Modified Fuzzy C-means (MFCM) for collaborative filtering (CF) problems. Though motivated by Fuzzy C-means (FCM), MFCM is designed to minimize Root Mean Squared Error of predictions directly. It also supplies a new explanation why matrix factorization usually works well for CF problems. We then develop two efficient algorithms — MFCM1 and MFCM2 to realize MFCM. Both of them acquire better predictions than FCM, and comparable accuracy with MF but better interpretability. Though MFCM proposed above is to cluster users, it is easy to generalize it to cluster movies or to cluster users and movies simultaneously.

On the other hand, we believe that there exist more powerful algorithms to solve (7) since MFCM1 and MFCM2 finally reduce the training RMSE to over 0.76 and 0.78 for the Netflix data set respectively. For furture work, we will explore some more efficient algorithms.

In another perspective, MFCM can be used to preprocess data in order to solve large-scale CF problems more efficiently. For example, the resulting probability matrix $\boldsymbol{Z}$ can be utilized to calculate similarity between users. Then the original neighbor-based methods can be used for prediction with much less computation.

6. ACKNOWLEDGMENTS

All the computations are mainly done with HP clusters in CCSE, Peking University. This work is partially supported by the China National Basic Research Program under the grant 2005CB321704.

7. REFERENCES

[1] R. Bell and Y. Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *Proc. IEEE International Conference on Data Mining (ICDM'07)*, 2007.

[2] J. Bennett and S. Lanning. The netflix prize. In *Proceedings of KDD Cup and Workshop*, 2007.

[3] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 43–52, 1998.

[4] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning*. Springer-Verlag, 2001.

[5] T. Hofmann, and J. Puzieha, Latent class models for collaborative filtering. In *Proc. 16th International Joint Conference on Artificial Intelligence*, pages 688–693, Morgan Kaufmann Publishers Inc., San Francisco, USA, 1999.

[6] T. Hong and D. Tsamis. Use of knn for the netflix prize. http://www.stanford.edu/class/cs229/proj2006/HongTsamis-KNNForNetflix.pdf.

[7] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *KDD-Cup and Workshop*. ACM Press, 2007.

[8] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted boltzmann machines for collaborative filtering. In *Proc. 24th Annual International Conference on Machine Learning*, 2007.

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

Improved Neighborhood-Based Algorithms for Large-Scale Recommender Systems

Andreas Töscher*
Technische Universitaet Graz
Inffeldgasse 16b
A-8010 Graz, Austria
toescher@sbox.tugraz.at

Michael Jahrer*
Technische Universitaet Graz
Inffeldgasse 16b
A-8010 Graz, Austria
jahrmich@sbox.tugraz.at

Robert Legenstein
Institute for Theoretical Computer Science
Technische Universitaet Graz
Inffeldgasse 16b
A-8010 Graz, Austria
legi@igi.tugraz.at

ABSTRACT

Neighborhood-based algorithms are frequently used modules of recommender systems. Usually, the choice of the similarity measure used for evaluation of neighborhood relationships is crucial for the success of such approaches. In this article we propose a way to calculate similarities by formulating a regression problem which enables us to extract the similarities from the data in a problem-specific way. Another popular approach for recommender systems is regularized matrix factorization (RMF). We present an algorithm – neighborhood-aware matrix factorization – which efficiently includes neighborhood information in a RMF model. This leads to increased prediction accuracy. The proposed methods are tested on the Netflix dataset.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: [Data mining, Recommender Systems, Collaborative Filtering, Netflix Competition]

General Terms

Latent factor model, Similarity matrix, Ensemble performance

Keywords

recommender systems, matrix factorization, KNN, Netflix, collaborative filtering

1. INTRODUCTION

Due to the increasing popularity of e-commerce, there is growing demand of algorithms that predict the interest of customers (called *users* in the following) in some product (called *item* in the following). Such interest is commonly quantified by a non negative number r which we call a *rating* in this article. Algorithms which predict a rating for each user-item pair are called recommender systems [1]. The predictions of recommender systems are in general based on a database which contains information about users and items. The *Collaborative Filtering* (CF) approach to recommender systems relies only on information about the behavior of users in the past. The pure CF approach is appealing because past user behavior can easily be recorded in web-based commercial applications and no additional information about items or users has to be gathered. CF algorithms for recommender systems are therefore easily portable.

More abstractly, the goal of CF is missing value estimation. Consider a system consisting of m users and n items. We define the set of users as the set of integers $\{1, \ldots, m\}$ and the set of items as the set of integers $\{1, \ldots, n\}$. The $m \times n$ rating matrix $\mathbf{R} = [r_{ui}]_{1 \le u \le m; 1 \le i \le n}$ stores the ratings of users for items where r_{ui} is the rating of user u for item i. The input to a CF algorithm is a set $\mathcal{L} = \{(u_1, i_1), \ldots, (u_L, i_L)\}$ of L user-item tuples referred to as *votes* and the corresponding ratings in the rating matrix. We assume that ratings in the rating matrix are non-zero if they are in the training set and zero if they are not in the training set, i.e., we assume $r_{ui} \neq 0$ if $(u, i) \in \mathcal{L}$ and $r_{ui} = 0$ otherwise. Such ratings not in the training set are called missing values. The goal of the system is to predict the missing values of R.

In fall 2006, the movie rental company Netflix started a competition, *the Netflix Prize*. The goal of the competition is to design a recommender system which improves on the Netflix recommender system *Cinematch* by 10% with regard to the root mean squared error (RMSE) on a published database. This database contains training data in the form of about 100 million ratings from about 480,000 users on 17,770 movies. Each rating in this database is an integer between 1 and 5. A probe set is provided which can be used to test algorithms. Furthermore, Netflix published a qualifying set which consists of user-item pairs but no ratings (the items correspond to movies in this database). The ranking of a submitted solution is based on this data set. The Netflix dataset captures the difficulties of large recommender systems. First, the dataset is huge and therefore the runtime and memory usage of potential algorithms become important factors. Second, the ranking matrix is very sparse with about 99 percent of its entries being missing such that many users have voted for just a few movies. The algorithms

*These authors contributed equally to this work.

2nd Netflix-KDD Workshop, August 24, 2008, Las Vegas, NV, USA.

presented in this article were tested on the Netflix dataset. However, their design is not specific to this dataset, thus the algorithms can be applied to other CF problems as well.

An obvious way to generate predictions of ratings is to calculate similarities between users and deduce a rating of some user u for an item i from ratings of that item by users with high similarity to user u. Similarly, a rating for some item i by user u can be predicted from ratings by that user for similar items. Such approaches have successfully been applied to the Netflix dataset. We deal with pure neighborhood-based approaches in Section 2. It turned out that they benefit from simple preprocessing where the ratings are first cleaned from so called *global effects* [2]. Global effects are linear relationships between the ratings and some simple variables like the time of the rating (which is provided in the Netflix dataset). We introduce in Section 2.1 four global effects which have not been considered before.

One problem of neighborhood-based approaches is the choice of the metric, or in other words, the measure of similarity between users or items.[1] A common way to measure the similarity between two users is the Pearson correlation between ratings of items which both users voted for. However, if two users have just a few common ratings (which is often the case in the Netflix dataset), the Pearson correlation is a bad estimate for their similarity. In Section 2.2 we propose a method where the similarities between items themselves are learned by gradient descent. Thus, the choice of a similarity measure is passed from the designer to the algorithm. One drawback of this method is the huge number of parameters which are fitted (the whole $n \times n$ matrix of item similarities has to be estimated) and thus its tendency to overfit the training data. This problem is tackled by a factorized version described in Section 2.3. This algorithm does not compute the whole similarity matrix but a low rank approximation in a linear latent factor model. This reduces the number of parameters significantly. A further advantage of factorized similarities is its memory efficiency. While the whole similarity matrix between users cannot be precomputed because of memory restrictions of computers to date, the online computation of correlations can be very time demanding. This makes naïve neighborhood-based approaches infeasible for large sets of elements like the set of users in the Netflix database. The factorized similarity model overcomes this problem. First, for a reasonable number of factors per user, the factor matrix can easily be held in memory, and second, for any two users the similarity is computed by just the inner product of two vectors. This model can also easily be extended to include information about unknown ratings (i.e., user-item pairs for which one knows that this user rated that item but the actual rating is unknown). The inclusion of unknown ratings in the prediction was first discussed in [7].

A regularized matrix factorization (RMF) produces a rank K approximation of the $m \times n$ rating matrix by factorizing it into a $m \times K$ matrix of user features and a $K \times n$ matrix of item features. RMF models are easy to implement and achieve good performance. Consequently, they are often used in CF algorithms. Overspecialization on the data points can be avoided by careful use of regularization techniques. We show in Section 3 that a hybrid approach which is partly neighborhood-based and RMF-based is a very effective CF method. This method performs slightly better than well-tuned RMF methods or restricted Boltzmann machines (RBMs). Additionally, the model can be trained very quickly.

[1]Obviously, one can also define a similarity measure for user-item pairs, an approach which is not discussed in this article.

Nb.	Side	Effect	RMSE
10	both	Previous effects	0.9659
11	item	average movie year	0.9635
12	user	movie production year	0.9623
13	user	STD of movie ratings	0.9611
14	item	STD of user ratings	0.9604

Table 1: Preprocessing for neighborhood models. The table shows the RMSE on the probe set of the Netflix dataset when accounting for 10 to 14 global effects (i.e., the row with Nb. i shows the error when one accounts for global effects 1 to i). The second column ("Side") specifies whether the effect defined in the third column ("Effect") needs the estimation of parameters for the users or for the items.

2. NEIGHBORHOOD-BASED MODELS

Neighborhood-based models for recommender systems commonly compute the similarity between users or items and use these similarities to predict unknown ratings. It is difficult to pre-calculate correlations between users for the Netflix database because the user correlation matrix can not be held in main memory of current computers. This makes the evaluation of naïve user-based neighborhood approaches very slow and therefore unpractical. We will discuss an efficient algorithm which is based on user similarities in Section 2.4. Before that, we discuss our algorithms for the case of item-based similarities and note that the principles apply to user-based similarities in the same way. The similarity c_{ij} between two items i and j is often estimated by the Pearson correlation between common ratings of that items, i.e., the correlation between the list of ratings of users $U(i,j) = \{u | (u,i) \in \mathcal{L} \text{ and } (u,j) \in \mathcal{L}\}$ which voted for both items i and j. The full neighborhood information is stored in the symmetric similarity matrix $\mathbf{C} = [c_{ij}]_{1 \le i,j \le n}$. Other similarity measures like the mean squared error (MSE) or a distance derived from the movie titles can also be used.

Algorithms in the flavor of the k-nearest neighbor (kNN) algorithm produce ratings based on the ratings of the k most similar items, i.e., the predicted rating $\hat{r}_{ui}$ of an user u for item i is computed as

$$\hat{r}_{ui} = \frac{\sum_{j \in N_k(u,i)} c_{ij} \, r_{uv}}{\sum_{j \in N_k(u,i)} c_{ij}}, \tag{1}$$

where $N_k(u,i)$ denotes the set of the k items most similar to item i that were rated by user u.

2.1 Preprocessing

In [2], so called "global effects" of the data were discussed and the removal of such effects from the data was proposed as an effective preprocessing step for the Netflix dataset. Such simple preprocessing turns out very useful if applied prior to kNN methods. As in [2], we model a global effect as a linear relationship between the ratings and some simple property of the votes. For each of the effects, the goal is to estimate one parameter per user or per item. For example, the effect may be the dependence of a vote (u,i) on the mean rating of user u. In this case, one would fit a parameter θ_i for each item such that $r_{ui} \approx \theta_i mean(u)$ The prediction is subtracted from the ratings and any further training is done on the residuals (see [2] for details).

We found four effects not described before which lower the RMSE on the probe set to 0.9604, see Table 1. The effects considered are the effect of the average production year of the movies the given user voted for ("average movie year"), the production year of the given movie ("movie production year"), the standard deviation of the ratings for the given movie ("STD of movie ratings") and the standard deviation of the ratings of the given user ("STD of user ratings").

The algorithms described below are tested for different preprocessings in order to facilitate comparison with other algorithms.

2.2 Regression on similarity

One problem of approaches based on the Pearson correlation between item ratings is that for many item pairs, there are only a few users which rated both items. For any two items i, j, we define the *support* s_{ij} for these items as the number of users which rated both items. The reliability of the estimated correlation grows with increasing support s_{ij}. For the Netflix dataset most correlations between movies are around 0. In order to decrease the influence of estimated correlations with low support on the prediction, we found it useful to weight correlations according to their support such that the similarity c_{ij} between item i and j is given by $c_{ij} = \tilde{c}_{ij} \frac{s_{ij}}{s_{ij}+\alpha}$ where $\tilde{c}_{ij}$ is the Pearson correlation between common ratings of the two items and α is a constant in the range of 100 to 1000 (this procedure was introduced in [3]).

In any case, the choice of the similarity measure is critical for the success of neighborhood-based algorithms. In this section we describe an alternative approach where the matrix of similarities $\mathbf{C}$ between items is learned by the algorithm itself. The matrix can be initialized with small random values around 0 or with Pearson correlations [2]. A prediction of the rating of user u for item i is calculated similar to equ. (1), but over the set $N(u,i) = \{j \neq i | (u,j) \in \mathcal{L}\}$ of all items different from i which user u voted for in the training set

$$\hat{r}_{ui} = \frac{\sum_{j \in N(u,i)} c_{ij} \, r_{uj}}{\sum_{j \in N(u,i)} |c_{ij}|}. \qquad (2)$$

Since similarities can become negative, we normalize by the sum of absolute similarities.

The objective function to minimize is given by the MSE with an additional regularization term

$$E(\mathbf{C}, \mathcal{L}) = \frac{1}{2} \sum_{(u,i) \in \mathcal{L}} (\hat{r}_{ui} - r_{ui})^2 + \gamma \sum_{j<k} c_{jk}^2,$$

where γ is a regularization constant. The model is trained by stochastic gradient descent on the objective function. For each training example we update only those similarities relevant for the example, i.e., for example (u,i) we update c_{ij} if $j \in N(u,i)$. The update of similarity c_{ij} is then given by

$$c_{ij}^{new} = c_{ij}^{old} - \eta \cdot \text{sign}\left((\hat{r}_{ui} - r_{ui}) \frac{\partial \hat{r}_{ui}}{\partial c_{ij}} \right) - \eta \, \gamma \, c_{ij}^{old}. \qquad (3)$$

[2]We used a uniform distribution in $[-0.1, 0.1]$ or Pearson correlations, with similar results.

Preprocessing	probe RMSE	qual. RMSE
Raw data	0.9574	0.9487
1GE	0.9458	0.9384
2GE	0.9459	0.9384
6GE	0.9372	0.9281
10GE	0.9349	0.9256
14GE	0.9331	0.9239

Table 2: The RMSE of the similarity regression model on the probe set (middle column) and the qualifying set (right column) of the Netflix dataset for preprocessings that accounted for 0 to 14 global effects (GE). For comparison, the Netflix recommender system achieves a RMSE of 0.9514 on the qualifying set. The probe RMSE for 10 and 14 global effects can be compared to the first and the last row of Table 1.

We opted to use the sign of the error gradient in (3) because this update turned out to be much more stable than the gradient itself. This choice was inspired by the sign-sign LMS rule (see, e.g., [5]). The number of trainable parameters in this model for the Netflix dataset is around 157 million which is very large. Hence training of the model is prone to early overfitting. Typical values of γ and η are 0.01. Results on the Netflix data are shown in Table 2. At a single epoch, approximately $L \cdot s_M$ similarities are updated, where L is the size of the training set $\mathcal{L}$ and s_M is the average number of votes per user (s_M is around 200 for the Netflix dataset). The training time for one epoch is in the range of one hour on a standard PC and usually a single epoch suffices.

2.3 Regression on factorized similarity

Gradient descent on the elements of the symmetric item similarity matrix $\mathbf{C}$ leads to early overfitting because of the huge number of trained parameters. In this section, we show that one can overcome this problem by learning a factorized version of $\mathbf{C}$. In other words, the algorithm learns two $K \times n$ matrices $\mathbf{P}$ and $\mathbf{Q}$ with $K \ll n$ and $\mathbf{C}$ is computed as

$$\mathbf{C} = \mathbf{P}^T \mathbf{Q}. \qquad (4)$$

Hence, we learn a rank-k approximation of $\mathbf{C}$ which drastically reduces the number of parameters. Only the upper triangle of $\mathbf{P}^T\mathbf{Q}$ is used to calculate similarities, since similarities are assumed to be symmetric, i.e., $c_{ij} = c_{ji}$. The similarity c_{ij} between items i and j is then given by

$$c_{ij} = \begin{cases} \mathbf{p}_i^T \mathbf{q}_j, & \text{if } i < j \\ \mathbf{p}_j^T \mathbf{q}_i, & \text{if } i > j, \end{cases} \qquad (5)$$

where $\mathbf{p}_i$ and $\mathbf{q}_i$ denote the ith column of $\mathbf{P}$ and $\mathbf{Q}$ respectively. Ratings are predicted as in the previous section by equ. (2).

The training schedule is similar to the direct similarity regression model with the difference that for every similarity c_{ij} (with $i < j$) we have to update the $2K$ parameters $p_{1i}, \ldots, p_{Ki}$ and $q_{1j}, \ldots, q_{Kj}$. Hence the training is slowed down by a factor of K as compared to direct similarity regression. Results on the Netflix data are shown in Table 3. The results are marginally better compared to the non-factorized version. Training the model took several hours on a standard PC.

Preprocessing	K	RMSE
10GE	80	0.9324
14GE	100	0.9313

Table 3: The RMSE of the factorized item similarity regression model on the probe set of the Netflix dataset for various preprocessings.

Preprocessing	K	RMSE
Raw	10	0.9951
2GE	10	0.9539
6GE	10	0.9469
14GE	20	0.9371

Table 4: The RMSE of the factorized user similarity regression model on the probe set of the Netflix dataset for various preprocessings.

2.4 Factorized user similarity matrix

An advantage of the factorized similarity model is that similarities between large sets of elements can be stored in memory. This enables us to store similarities between users in the factor matrices. In order to learn user similarities we perform gradient descent on the factorized user similarity matrix. All calculations and update rules are mirrored versions of those discussed above for the item similarity matrix. For the Netflix dataset the training time increases by a factor of 30 compared to the training time of the factorized item similarity model since there are approximately 30 times more users than items in the database.

The results of the model for a few different preprocessings of the data are shown in Table 4.[3] Although the results are similar the those of the factorized item similarity model, the algorithm is still useful since the information extracted from user similarities is different from that when item similarities are used. This contributes to the performance if the models are finally combined for a single prediction, see Section 4.

2.5 Incorporating unknown ratings

The model can be extended to include unknown ratings. This helps in general on users with few ratings in the training set. Let $\mathcal{L}'$ denote the set of votes for which the rating is unknown (for the Netflix dataset, our set $\mathcal{L}'$ consisted of votes in the probe set as well as those in the qualifying set). Let $N'(u,i) = \{j \neq i | (u,j) \in \mathcal{L}'\}$ denote the set of items different from i user u has voted for with unknown rating. Then, the prediction $\hat{r}_{ui}$ of a rating for user u on item i is given by

$$\hat{r}_{ui} = \frac{\sum_{j \in N(u,i)} c_{ij}\, r_{uj} + \sum_{j \in N'(u,i)} c_{ij}\, \tilde{r}_{uj}}{\sum_{j \in N(u,i)} |c_{ij}| + \sum_{j \in N'(u,i)} |c_{ij}|}, \quad (6)$$

where $\tilde{r}_{uj}$ are estimates of the unknown ratings (they are parameters of the model which are trained, see below). Training of the similarities is done as in the basic model (see equ. (3)) with the difference that for training example (u,i) we update all c_{ij} for $j \in N(u,i) \cup N'(u,i)$. The unknown ratings $\tilde{r}_{uj}$ are trained simultaneously with gradient descent.

[3]Because of the time demands of this algorithm, training was stopped after the presentation of only 30% of the training set.

One can initialize each unknown rating with the mean rating of the corresponding item. Slightly better performance can be obtained if one initializes the unknown ratings with predictions of a neighborhood-based approach, see equ. (1) (the reported results were obtained in this manner). On the Netflix dataset, this model achieved a RMSE of 0.9278 on the probe set with a preprocessing that accounted for 14 global effects. This is an improvement of 0.053 over the model without unknown ratings (see Table 2).

3. NEIGHBORHOOD-AWARE MATRIX FACTORIZATION

In this section we present an algorithm – neighborhood-aware matrix factorization (NAMF) – which efficiently incorporates a linear regularized matrix factorization (RMF) in a neighborhood-based model. More specifically, for a given vote (u,i), the algorithm computes three predictions: a prediction $\hat{r}_{ui}^{MF}$ which is based on a RMF, a prediction $\hat{r}_{ui}^{user}$ is based on a user-neighborhood model, and a prediction $\hat{r}_{ui}^{item}$ which is based on a item-neighborhood model. Both neighborhood-based models utilize predictions from the RMF model if needed. The final prediction of the algorithm is a combination of the three predictions.

3.1 Regularized matrix factorization model

A RMF computes a rank K approximation $\mathbf{R}' = \mathbf{A}\mathbf{B}^T$ of the rating matrix $\mathbf{R}$, where $\mathbf{A} \in \mathbb{R}^{m \times K}$ is the user factor matrix and $\mathbf{B} \in \mathbb{R}^{n \times K}$ is the item factor matrix. The entries of these matrices are determined such that $r_{ui} \approx r'_{ui}$ for all votes $(u,i) \in \mathcal{L}$. After the factor matrices $\mathbf{A}$ and $\mathbf{B}$ have been determined by the training algorithm, the prediction $\hat{r}_{ui}^{MF}$ for a vote (u,i) is given by $\hat{r}_{ui}^{MF} = r'_{ui} = \sum_{k=1}^{K} a_{uk} b_{ik}$. Because the rating matrix is usually sparse, additional regularization is needed. Using a regularization as proposed in [9], [6], [8] leads to the error function

$$E(\mathbf{A}, \mathbf{B}, \mathcal{L}) = \sum_{(u,i) \in \mathcal{L}} (r_{ui} - \hat{r}_{ui}^{MF})^2 + \frac{\lambda}{2}(\|\mathbf{A}\|_F^2 + \|\mathbf{B}\|_F^2), \quad (7)$$

where $|| \cdot ||_F$ denotes the Frobenius norm and λ is the regularization parameter. We use stochastic gradient descent to minimize this error function. The update equations for a training example (u,i) are therefore

$$a_{uk}^{new} = a_{uk}^{old} + \eta \cdot (e_{ui} b_{ik}^{old} - \lambda a_{uk}^{old}) \quad (8)$$

$$b_{ik}^{new} = b_{ik}^{old} + \eta \cdot (e_{ui} a_{uk}^{old} - \lambda b_{ik}^{old}), \quad (9)$$

for $k = 1, \ldots, K$ and

$$e_{ui} = r_{ui} - \sum_{k=1}^{K} a_{uk}^{old} b_{ik}^{old}. \quad (10)$$

3.2 User-neighborhood model

The similarity of two users can be measured by the Pearson correlation $\tilde{\rho}_{uv}^{user}$ between the list of ratings for items which were rated by both users. In order to decrease the influence of correlations with low support we shrink each correlation according to their support s_{uv} [3]:

$$\rho_{uv}^{user} = \frac{s_{uv} \tilde{\rho}_{uv}^{user}}{s_{uv} + \alpha^{user}}, \quad (11)$$

where the parameter α^{user} is determined as discussed below (in order to facilitate readability we denote all variables and parameters of the user-neighborhood model by a "user" superscript and those of the item-neighborhood model by a "item" superscript). However, the use of the Pearson correlation may be problematic in some cases. The most severe problem occurs if the number of common ratings is small for most user pairs (i.e., the support for most user pairs is low). In this case, the Pearson correlation between these ratings is a very unreliable measure of similarity. For many datasets however there exist for most users other users such that the number of common rated items is large, and reliable correlations can be calculated for these pairs. We will make use of this observation below. Another problem of employing correlations between users is the size of the correlation matrix. For the Netflix dataset where the number of users is around 480,000, the whole matrix needs about one TByte of memory. We overcome this problem by storing for each user u only the correlations with the J users with highest correlation to u. A rating prediction $\hat{r}_{ui}^{user}$ is then computed as the weighted sum over the ratings of these best correlating users where the rating r_{vi} is given by the predicted rating of the RMF model or a rating from a training example if it exists

$$\hat{r}_{ui}^{user} = \frac{\sum_{v \in U_J(u)} c_{uv}^{user} r_{vi}}{\sum_{v \in U_J(u)} c_{uv}^{user}}, \tag{12}$$

where $U_J(u)$ denotes the set of J users with highest correlation to u. Each weighting coefficient c_{uv}^{user} is computed from the Pearson correlation ρ_{uv}^{user} by applying a squashing function

$$c_{uv}^{user} = \left(\sigma\left(s^{user} \rho_{uv}^{user} - b^{user}\right)\right)^{\gamma^{user}}, \tag{13}$$

where the scaling factor s^{user}, the bias b^{user}, and the exponent γ^{user} are global parameters which were determined as described below. The sigmodial squashing function $\sigma(\cdot)$ is given by

$$\sigma(x) = \frac{1}{(1 + \exp(-x))}. \tag{14}$$

3.3 Item-neighborhood model

For the item side the same principle can be applied. Correlations $\tilde{\rho}_{ij}^{item}$ between common rated items are shrunk

$$\rho_{ij}^{item} = \frac{s_{ij} \tilde{\rho}_{ij}^{item}}{s_{ij} + \alpha^{item}}. \tag{15}$$

For each item i only the correlations with the J items with highest correlation to i are stored. A rating prediction is then computed as the weighted sum over the ratings of these best correlating items $I_J(i)$. The weighting coefficients are given by

$$c_{ij}^{item} = \left(\sigma\left(s^{item} \rho_{ij}^{item} - b^{item}\right)\right)^{\gamma^{item}}, \tag{16}$$

where ρ_{ij}^{item} denotes the Pearson correlation between the ratings of users that rated both items i and j, and α^{item}, s^{item}, b^{item}, and γ^{item} are constants. The rating prediction $\hat{r}_{ui}^{item}$ for a vote (u, i) is given by

$$\hat{r}_{ui}^{item} = \frac{\sum_{j \in I_J(i)} c_{ij}^{item} r_{uj}}{\sum_{j \in I_J(i)} c_{ij}^{item}}. \tag{17}$$

3.4 Combining the information

The predictions from the RMF model, the user neighborhood model and the item neighborhood model are combined in a single rating. The obvious way to archive this is an optimal linear combination of the three predictions. Experiments have shown that the predictive accuracy of the models strongly depends on the support and the number of ratings from the training data (as opposed to those from the RMF model) used in the neighborhood models. So we use a weighted sum, based on this information to combine the predictions:

$$\hat{r}_{ui} = \frac{\tilde{S}(u,i)^{\delta} \cdot \hat{r}_{ui}^{MF} + \hat{\beta}\hat{S}(u,i)^{\hat{\delta}} \cdot \hat{r}_{ui}^{user} + \bar{\beta}\bar{S}(u,i)^{\bar{\delta}} \cdot \hat{r}_{ui}^{item}}{\tilde{S}(u,i)^{\delta} + \hat{\beta}\hat{S}(u,i)^{\hat{\delta}} + \bar{\beta}\bar{S}(u,i)^{\bar{\delta}}} \tag{18}$$

$$\tilde{S}(u,i) = min\{N_u, N_i\}. \tag{19}$$

In the equation above, $N_u = |\{i | (u,i) \in \mathcal{L}\}|$ denotes the number of votes of user u, and $N_i = |\{u | (u,i) \in \mathcal{L}\}|$ denotes the number of votings for item i. $\bar{S}(u,i) = |\{v \in U_J(u) | (v,i) \in \mathcal{L}\}|$ and $\hat{S}(u,i) = |\{j \in I_J(i) | (u,j) \in \mathcal{L}\}|$ denote the number of votes from the training set used to calculate the corresponding ratings.

The training schedule can be summarized as follows. First, correlations $\tilde{\rho}_{uv}^{user}$ between users and correlations $\tilde{\rho}_{ij}^{item}$ between items are computed. The best correlating users/items are computed according to the shrunken correlations (we used $\alpha^{user} = 10$ for user correlations and for item correlations $\alpha^{item} = 30$) and the corresponding correlations (not shrunk) are stored. This step is the computationally most demanding one. Then the RMF is computed. Once this is done, the predictions of the neighborhood models can be computed very efficiently. Then, good values for the 13 constants α^{user}, α^{item}, $\bar{\beta}$, $\hat{\beta}$, s^{user}, s^{item}, b^{user}, b^{item}, γ^{user}, γ^{item}, δ, $\bar{\delta}$, and $\hat{\delta}$ in the model are determined with a genetic algorithm. Because the evaluation of individuals is very fast, this optimization step can be done quite efficiently (on a standard PC this step needed 1-2 hours). Once the model is trained, predictions can be generated very quickly.

3.5 Experimental results

The RMF model was trained on the residuals of the first global effect (movie effect) described in [2]. The use of this effect slightly improves RMF performance whereas the use of all global effects decreases the performance of the RMF model. All RMF models were trained with stochastic gradient descent using $\eta = 0.002$ and $\lambda = 0.02$. The weights were initialized to small values sampled from a normal distribution with zero mean and standard deviation 0.001. The neighborhood models were trained on preprocessed data that incorporated 10 global effects. The results are shown in Table 5. The time to train the whole model for $K = 600$ features was about 24 hours on a standard PC.

In comparison, a restricted Boltzmann machine on the Netflix probe data achieved a RMSE of 0.907 (see Fig. 4 in [7]). Another approach which combines a neighborhood model with a RMF was described in [2]. This algorithm obtained a RMSE of 0.9071 on the Netflix probe set. Also, a matrix factorization where the features were trained with respect to some neighborhood relation was outlined in [8]. However, this method was only used for data visualization.

Features K of the RMF	RMSE
10	0.9175
50	0.9069
100	0.9056
300	0.9046
600	0.9042

Table 5: RMSE of different neighborhood-aware matrix factorizations on the Netflix probe data. Preprocessing for the neighborhood model was done on 10 global effects, the neighborhood size was J =50.

4. ENSEMBLE PERFORMANCE

The final goal of each team that participates in the Netflix contest is the prediction of unknown ratings with optimal accuracy. In order to achieve maximal prediction accuracy, it is a common strategy to combine predictions of different algorithms into a final one. We did linear blending on the probe set, which was not used for training, similar to [4]. Whether an algorithm is particularly powerful on a given data point (u, i) depends strongly on the support of the vote, i.e., the number of votes of user u and the number of votes for item i. Consequently, a linear combination of predictions for data points with low support will be quite different from a linear combination for data points with high support. We therefore divided the probe set into *slots* based on the support of the data points. To obtain a single value from the user support and the item support we combined them by taking the minimum of both. This procedure is called "slot blending" [4]. The slot boundaries were chosen such that the number of ratings in the slots was approximately uniform.

For each slot, the final prediction is then computed as a linear combination of the predictions of the individual algorithms. Suppose one wants to combine the predictions of N algorithms. These predictions are first stored in a predictor matrix $\mathbf{P} \in \mathbb{R}^{l \times N}$ where l is the number of votes in the slot and p_{ij} is prediction of algorithm j for the i-th vote in the slot. The interpolation weights $\mathbf{w}$ are computed with the pseudo-inverse of P as $\mathbf{w} = (\mathbf{P}^T\mathbf{P})^{-1}\mathbf{P}^T\mathbf{q}$ where $\mathbf{q}$ is the column vector of probe ratings of the slot. The final prediction for a vote from the qualifying set which falls – according to its support – into this slot is then given by the linear combination of the individual predictions with the interpolation weights $\mathbf{w}$.

Using this method, we calculated the ensemble performance of the algorithms proposed in this article. The RMSE of the ensemble on the probe set was 0.8981 which results in a RMSE of 0.8919 on the qualifying set (for predictors which were re-trained after blending with the probe ratings included). This is an improvement of 6.25% over the Cinematch system. The proposed methods can well be combined with other powerful algorithms like different kinds of matrix factorizations and restricted Boltzmann machines to further improve prediction accuracy.

5. CONCLUSIONS

In this article, we proposed several neighborhood-based algorithms for large-scale recommender systems. An important property of these algorithms is that their memory usage scales linearly with the number of users or items as compared to a quadratic scaling of most other neighborhood-based approaches. This makes the algorithms scalable to large-scale problems. To date it seems that powerful solutions for collaborative filtering problems need to combine the predictions of a diverse set of single algorithms. This procedure is able to combine the specific advantages of single algorithms. The standard approach is linear blending, where the predictions are simply combined in a linear way after training. The neighborhood-aware matrix factorization algorithm tries to combine the advantages of two powerful methods – a RMF approach and neighborhood-based approach – in a more direct way: The predictions of one algorithm are used to estimate unknown variables in the other ones. One can therefore hope that the combination of them is more than the (weighted) sum of its parts. Such hybrid models are promising candidates for future research.

6. ACKNOWLEDGMENTS

We thank Netflix for providing this nice dataset. We also thank the participants of the previous KDD workshop in 2007 for providing their inspiring ideas in the field of recommender algorithms. Also thanks to the active Netflix community for discussing all kind of things regarding various implementation details on proposed algorithms. R. L. was partially supported by project # FP7-216886 (PASCAL2) of the European Union.

7. REFERENCES

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17(6):734–749, 2005.

[2] R. Bell and Y. Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *IEEE International Conference on Data Mining*. KDD-Cup07, 2007.

[3] R. Bell, Y. Koren, and C. Volinsky. Modeling relationships at multiple scales to improve accuracy of large recommender systems. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 95–104, New York, NY, USA, 2007. ACM.

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

[5] S. Dasgupta, C. R. Johnson, and A. M. Baksho. Sign-sign LMS convergence with independent stochastic inputs. *IEEE Transactions on Information Theory*, 36(1):197–201, 1990.

[6] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. *Proceedings of KDD Cup and Workshop*, 2007.

[7] R. Salakhutdinov, A. Mnih, and G. E. Hinton. Restricted boltzmann machines for collaborative filtering. In *ICML*, pages 791–798, 2007.

[8] G. Takács, I. Pilászy, B. Németh, and D. Tikk. On the gravity recommendation system. In *KDD Cup Workshop at SIGKDD 07*, pages 22–30, August 2007.

[9] M. Wu. Collaborative filtering via ensembles of matrix factorizations. *Proceedings of KDD Cup and Workshop*, 2007.