Proceedings of the Second KDD Workshop on

Large-Scale Recommender Systems and the Netflix Prize Competition

August 24, 2008, Las Vegas, Nevada, USA

held in conjunction with

SIGKDD'08

Workshop Organizers

Alex Tuzhilin (chair)
Yehuda Koren (co-chair)
Jim Bennett
Charles Elkan
Daniel Lemire

Foreword

In October 2006, Netflix released a large dataset and launched the Netflix Prize Competition that attracted over 20,000 participants from 167 countries. As a part of this competition, many interesting data mining recommendation techniques have been developed that, by the very nature of the Netflix dataset, should be scalable to large recommendation problems.

This workshop follows on the previously held KDD 2007 workshop with the goal to exchange the ideas and present and discuss novel recommendation methods that specifically focus on the scalability and performance issues pertaining to large-scale datasets for recommender systems and the Netflix competition in particular.

The call for papers for the workshop attracted 10 submissions, out of which 5 papers were selected for the presentation at the workshop among which 3 papers were by the authors of leading teams in the Netflix Prize competition. The workshop is opened with the keynote address on "Exploring User Opinions in Recommender Systems" by Bing Liu from the University of Illinois who provides his views on future developments of recommender systems based on opinion mining and sentiment analysis and sets a stage for the subsequent presentations and discussions among the participants.

The workshop organizers would like to thank the authors of the submitted papers, the members of the Program Committee for providing careful reviews of the submissions and Netflix for launching this competition, without which this workshop would not have been possible.

Finally, we wish all the participants intellectually simulating and productive discussions during the workshop. We do hope that they would lead to further advances of the field of recommender systems and to new insights and developments of novel or improved approaches to the solutions of the Netflix problem.

Jim Bennett, Netflix.
Charles Elkan, University of California, San Diego
Yehuda Koren (co-chair), AT&T Labs--Research
Daniel Lemire, University of Quebec at Montreal (UQAM)
Alex Tuzhilin (chair), NYU Stern,
Workshop Organizers

Program committee

- Gedas Adomavicius, University of Minnesota
- Deepak Agarwal, Yahoo! Research
- Shlomo Berkovsky, University of Melbourne
- Robin Burke, DePaul University
- Thomas Hofmann, Google
- Kartik Hosanagar, The Wharton School, University of Pennsylvania
- Zan Huang, Penn State
- George Karypis, University of Minnesota
- Ramayya Krishnan, CMU
- Bing Liu, University of Illinois, Chicago
- Ben Marlin, University of Toronto
- Bamshad Mobasher, DePaul University
- Naren Ramakrishnan, Virginia Tech
- John Riedl, University of Minnesota
- Padhraic Smyth, University of California, Irvine

Keynote:

Exploring User Opinions in Recommender Systems

Bing Liu

University of Illinois at Chicago

Abstract

Traditionally, recommender systems operate based on user-behavior and rating data at the personal and/or aggregate level. In this talk, I will try to go beyond this tradition to discuss some new/future developments of recommender systems, even general advertising systems for that matter, based on opinions on the Web (e.g., in reviews, forum discussions, blogs, etc). Recommendations based on such data can be highly targeted and can also be embedded widely in the most appropriate context. Needless to say, I will introduce some recent developments in the area of opinion mining and sentiment analysis, and discuss whether these developments are ready for prime time.

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, $\mathcal{P} = \{(u,m)|r_{u,m} \text{ is in the training set}\}$. 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?

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

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

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

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

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

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

(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:

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

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:

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

(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 $\bar{r}$:

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

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

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

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,

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

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:

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

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.

Putting the collaborator back into collaborative filtering

Gavin Potter
Lawford Rd
London, NW5 2LH
England
+ 44 7710 297631
g.potter@btconnect.com

ABSTRACT

Most of the published approaches to collaborative filtering and recommender systems concentrate on mathematical approaches for identifying user / item preferences. This paper demonstrates that by considering the psychological decision making processes that are being undertaken by the users of the system it is possible to achieve a significant improvement in results. This approach is applied to the Netflix dataset and it is demonstrated that it is possible to achieve a score better than the Cinematch score set at the beginning of the Netflix competition without even considering individual preferences for individual movies. The result has important implications for both the design and the analysis of the data from collaborative filtering systems.

Categories and Subject Descriptors

J4 [**Social and Behavioral Sciences**]: Psychology

General Terms

Algorithms, Human Factors,

Keywords

Psychology, Behavioral Economics, recommender systems, collaborative filtering.

1. INTRODUCTION

Collaborative filtering techniques are increasingly being used to make personalized recommendations to users based on their perceived tastes and preferences. These tastes and preferences are often elicited by asking the user to score items on a scale (typically, but not always from 1 to 5). From these scores the underlying preferences are inferred by considering the interdependencies between different users and different products based on these scores and then these preferences are used to recommend items.

Conference 2nd Netflix-KDD workshop, August 24th,2008,Las Vegas,NV,USA

It is important to note that when a user is using such a system, the user is being asked to perform two separate tasks. First they are being asked to estimate their preference for a particular item and secondly they are being asked to translate that preference into a score on whatever rating scale is being used. There is a significant issue with this approach in that the scoring system, therefore, only produces an indirect estimate of the true preferences of the user. In mathematical terms this can be expressed as:

$$\text{Score} = f_{user}(\text{item preference}_{user}) \quad (1)$$

Where $f_{user}()$ is some, as yet, unspecified scoring function that translates individual item preferences into a score on the rating scale being used and maybe different for different users.

If the users of the system are using different scoring functions, then in order to make true comparisons between the underlying preferences of different users, it is first necessary to normalize the effects of the different scoring systems being used by the different users of the system.

In October 2006, the online movie rental company, Netflix announced a contest to predict movie ratings. To achieve this Netflix released a comprehensive dataset of more than 100 million ratings made by 480,000 customers on 17,770 movies. We are indebted to Netflix for releasing such a large dataset of clean data. This dataset is almost undoubtly the largest dataset of repeated decisions by individuals available publicly and provides a rich set of data for the analysis of human decision making.

This paper:

- Examines whether different users use different scoring functions
- outlines a number of approaches for normalizing the effects of the different scoring functions being used by individual users of the system,
- demonstrates the impact that the normalization of the scoring function can achieve, and
- discusses the implications of the findings for the future development of collaborative filtering and rating systems.

2. Do different users use different scoring functions?

Apriori, it seems very unlikely that every user utilises exactly the same scoring function when translating an underlying preference for an item into a score on a rating scale, especially given the fact that detailed instructions for the conversion of underlying preferences into numerical ratings are rarely given to users of collaborative filtering and recommendation systems.

To investigate whether this was true with the Netflix recommendation system, we first analysed the mean score for each user who had more than 100 ratings on the database. The distribution of the user means are shown below.

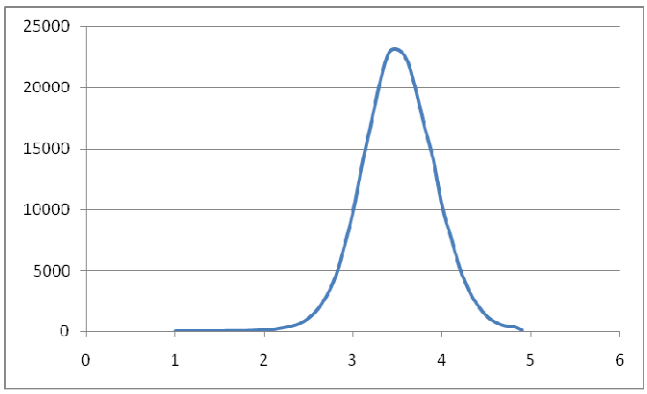

Figure 1 – Distribution of mean scores by user

As can be seen, the mean scores by user range from 1 to 4.9, i.e. at least one user rated all the movies rated a 1 and at least one user gave an average score of 4.9 out of 5. Even excluding outliers Figure 1 demonstrates that there is a wide dispersion around the average score.

It is, impossible, using such a simple analysis, to say with certainty that the differences are due to the scoring function being used, as it is possible that an individual has only seen films that they dislike or like. However, discussions with users of rating scales suggest anecdotally that they do use the rating system in different ways – some reserving the highest score only for films that they regard as truly exceptional, others using the score for films that they simply enjoy. Indeed some Netflix users (personal correspondence) go to considerable trouble to calibrate their scores with others. One group of users, for example, determined to rank all movies on the Netflix system, joined together and agreed a specific meaning for each of the scores that were available to them before starting the scoring exercise. We, therefore, believe that at least some of the variance of the mean scores can be attributed to users using the scoring scale in different ways.

Similarly, we examined the variance of the scores used by each user, and the results are shown below

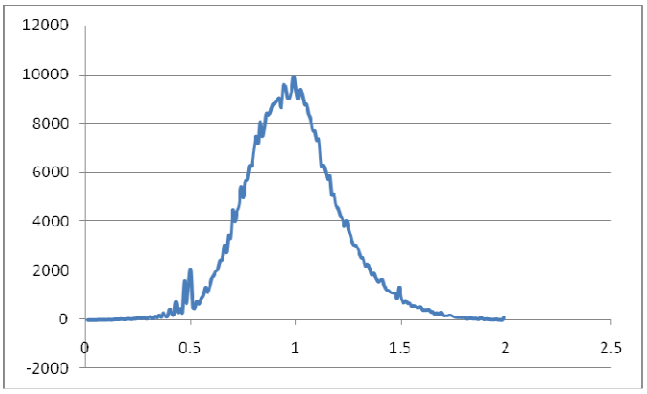

Figure 2 – Distribution of variances of scores by customer

Again, it is possible, that these scores represent a consistent translation of the underlying preferences between the users of the system for the movies that they have watched, some users having only small differences in preferences of the films they have rated, and others having large differences in preferences, but the large range of the variance of scores used by different users does potentially imply that different users are translating their preferences into scores using different scoring functions, with some users using a much narrower range of scores than others.

3. Developing a scoring function

3.1 Basic scoring Function

There are an infinite number of possible scoring functions that could be used, and without an explicit explanation from each user it is impossible to be exact about the function that might be used.

To derive a set of possible candidate functions for examination and testing, we therefore reversed the problem and considered what data we had available from which we could derive such functions.

The Netflix dataset contains triplets comprising of an id for the movie, an id for the customer and the date on which the rating was made. This suggests that, at the most abstract level, a starting point that does not require additional data collection or experimentation would be to examine functions of the form.

$$\text{Score} = f_{user,movie,date}(\text{preference}_{user,movie,date}) \quad (2)$$

Where f is a scoring function and preference is a preference function both depending on the user, the movie and the date.

There are still, of course, an infinite number of functions that meet the overall structure described in equation 2. We rely heavily on introspection and discussion with rating scale users to narrow the candidate set of functions to be examined.

To test the impact of applying different candidate scoring functions, we first derived a very simple preference function. The preference function that we used was the mean score for a

particular movie across the whole population. i.e. we assumed that different films would be preferred in different amounts, but they would be preferred equally by everyone in the population. When this function was used on its own, the score achieved on the probe data set was an rmse of 1.0527.

The initial and simplest scoring function that we explored was simply to assume that the score consisted of two components:

$$Score_{movie,customer} = preference_{movie} + bias_{customer} \quad (3)$$

A similar function has already been explored by Bell and Koren [1]. Their strategy was to consider each effect separately and then to subtract that effect from the score and to calculate the next effect on the remaining residual. So with the above function, they first calculated a movie preference by taking the mean score for each movie. They then subtracted it from the original rating and calculated a customer bias based on taking the mean of the residuals by customer. Their results were further enhanced by the use of shrinkage factors, whereby only a proportion of the biases were included depending on the number of datapoints used to calculate them.

The approach we took was different. The mean score for a film, may, for example, be influenced by those who have watched it. So if a film is rated mainly by users who tend to give low scores then the mean for the movie will be understated when compared to the mean that would be achieved if the whole population had rated the movie. Similarly, a customer who had only watched poor quality (low preference) films will have an average score that is lower than the average score that would be achieved if they had watched all the films on the database (and vice versa).

The standard approach within psychology for examining such effects would be to use factor analysis. However, given the number of factors that are involved, (480,000 customer factors and 17,770 movie factors), despite the simplicity of the function it is not possible to solve for both customer bias and movie preference simultaneously. We tried a number of approaches to solving the equations including gradient descent, but found that the most effective both in computer time and in the accuracy of results was simply to iterate around the equation. So we first calculated the preference $_{movie}$ by taking the mean score per movie and then calculated the bias $_{customer}$. by taking the mean score of the residuals once the $preference_{movie}$ had been calculated. We then looped around these equations until the change in the score achieved on the training values fell below a small threshold value.

To test whether this led to an improvement, we use the published results from [1] as a comparison. The results are shown in Table 1 below. Including the customer bias does lead to a substantial improvement in scores, bringing the score on the probe dataset down to an rmse of 0.9824.

In pseudocode, the algorithm works as follows.

Set movie preference = 0 : customer bias = 0

Loop

- Subtract customer bias
- Add back movie preference
- Calculate movie preference
- Subtract movie preference
- Add back customer bias
- Calculate customer bias
- Calculate training score

End loop when change in score < threshold.

3.2 Including the range of scores used by a user

Figure 2 suggests that the results might also be improved if we could take into account the range of the scores used by a user. To account for the range of scores being used we assumed that the score for each user,movie pair was a result of three factors, a movie preference, a customer preference, and the variance of scores used by the user as shown in (4) below.

$$Score = (preference_{movie} + bias_{user}) * (variance_{user})^{0.5} \quad (4)$$

To calculate the parameters in this model, we first divided each users' scores by the standard deviation of the scores of the user and then repeated the iterative algorithm. We then reversed the process and multiplied the calculated movie preferences and customer biases by the standard deviation of each user's scores to derive the final scores. The results are shown below.

Table 1. Scores of simple bias model

Approach	Score*
BellKor [1]	.9841
Iterative algorithm	.9824
Iterative algorithm plus variance adjustment	.9808

*Note all scores are reported on the probe dataset with the probe data removed from the training data. The scores reported are the RMSE scores.

Again, an improvement is achieved, suggesting that the incorporation of a scoring function calibrated for an individual user can lead to an improvement in results. It is interesting to note that these scores were achieved without the fitting of any shrinkage factors.

3.3 Adding other effects

We have considered a wide range of other models. One of the simplest and most powerful that we found was to include the impact of the date of the rating. It seems intuitively plausible that

a user would allocate different scores depending on the mood that they were in on the date of the rating. It also seems unlikely that an individual user would apply their scoring function in exactly the same way at each date. The Netflix dataset contains a considerable number of occasions where a user has rated more than one movie at a time, so it is possible to extract from the dataset an average score on a particular day. In this case the function that we fitted was:

$$\text{Score} = (\text{preference}_{user} + \text{bias}_{customer} + \text{bias}_{day}) * (\text{variance}_{user})^{0.5}$$

In this case we found that the simple iterative approach did not work as well as using gradient descent. To achieve this we differentiated the function .

$$\text{Error} = (\text{Score}/\text{variance}_{user} - \text{preference} - \text{biases})^2 \quad (6)$$

by each of the biases and then iterated through each score and adjusted each of the biases according to the formula:

$$\text{bias} = \text{bias} + \text{lrate} * \text{difference} \quad (7)$$

where lrate = the learning rate typically 0.001

$$\text{difference} = \text{Score}/\text{variance}_{user} - \text{biases}$$

Although we did not introduce any regularization into the algorithm to control the size of the biases we did find that the results could be improved if a different learning rate was used for the different biases. We experimented with a variety of different learning rates and found that the ones that worked best were of the form:

$$\text{Lrate} = \text{lrate} * n_{cust} / (n_{cust} + \quad) \quad (8)$$

Where n = the number of movies scored by a particular customer.

This was used for the customer bias and similar learning rates were used for the movie preferences and the day bias. When we calculated these biases using a variable learning rate we achieved a score of 0.9675 rmse on the probe dataset.

3.4 Adding the effects together

We have used this approach to simultaneously examine many such effects and have achieved a variety of different scores. For comparison purposes we have included our results on the above effects together with the global effects identified by [1]. These results are shown below.

When the probe data is included within the training data and a submission made to the competition, the result is 0.9488. This score comfortably exceeds the score achieved by the Cinematch algorithm at the beginning of the competition and is achieved without any consideration of how much an individual user likes or dislikes an individual movie.

Table 2 – Results of Bellkor vs gradient descent algorithm

Effect	Bellkor*	New algorithm
Overall mean	1.1296	N/a
Movie effect	1.0527	1.0527
User bias	.9841	.9808
Day bias	N/a	.9675
User * time(user) ^ .5	.9809	.9680
User * Time (movie) ^.5	.9786	.9667
Movie * Time (movie) ^ .5	.9767	.9653
Movie * Time (user) ^.5	.9759	.9643
User * Movie average	.9719	.9598
User * Movie support	.9690	.9582
Movie * User average	.9670	.9561
Movie * User support	.9657	.9552

For a description of the global effects see [1].

4. Summary

The above shows that the introduction of even simple individual scoring functions for users can have a significant impact on the overall results achieved. Of more importance, developers of collaborative filtering and recommendation systems need to recognize that their choice of scale and design of their system will introduce noise that will mask the real preferences of the individual.

It is our view that when it is important to extract exact preference information and preference orderings between items, then the designers of such systems should consider other options such as the paired comparison of items. These types of system have a significant advantage in that they only require users to provide preference ordering information and the users do not, therefore, have to translate their preferences into a scale. Because one can also examine the transitivity of the results these systems provide an inbuilt estimate of the reliability of the results. The disadvantage, of course, is a more complex user interface and users may not be prepared to respond to such types of system.

Decision making biases can be identified in the Netflix dataset and incorporation of these biases into a scoring function can significantly improve results. Much more work needs to be done to understand further these biases and how they can be taken into account when designing and analyzing collaborative filtering and recommendation systems.

5. REFERENCES

1. R.Bell and Y.Koren, “Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights”, IEEE International Conference on Data Mining (ICDM’07), IEEE, 2007.

From hits to niches? or how popular artists can bias music recommendation and discovery

Òscar Celma
Music Technology Group
Universitat Pompeu Fabra
Barcelona, SPAIN
oscar.celma@iua.upf.edu

Pedro Cano
Barcelona Music and Audio Technologies
Llacuna 162, 08018
Barcelona, SPAIN
pedro.com

ABSTRACT

This paper presents some experiments to analyse the popularity effect in music recommendation. Popularity is measured in terms of total playcounts, and the Long Tail model is used in order to rank music artists. Furthermore, metrics derived from complex network analysis are used to detect the influence of the most popular artists in the network of similar artists.

The results from the experiments reveal that—as expected by its inherent social component—the collaborative filtering approach is prone to popularity bias. This has some consequences on the discovery ratio as well as in the navigation through the Long Tail. On the other hand, in both audio content–based and human expert–based approaches artists are linked independently of their popularity. This allows one to navigate from a mainstream artist to a Long Tail artist in just two or three clicks.

Categories and Subject Descriptors

H3.3 [**Information Search and Retrieval**]: Information filtering, Selection process; G.2.2 [**Graph Theory**]: Graph algorithms

Keywords

recommender systems, popularity, long tail, evaluation, complex network analysis

1. INTRODUCTION

The Long Tail is composed by a very few popular items, the well–known *hits*, and the rest, located in the heavy tail, that does not sell *that well* [1]. The Long Tail offers the possibility to explore and discover—using automatic tools; such as recommenders—from vast amounts of data. Until now, the world was ruled by the *Hit or Miss* classification, due in part to the shelf space limitation of the brick–and–mortar stores. A world where a music band could only succeed selling millions of albums, and touring worldwide. Nowadays, we are moving towards the *Hit vs. Niche* idea, where there is a large enough availability of choice to satisfy even the most 'Progressive–obscure–Swedish–metal" fan. The problem, though, is to filter and present the *right* artists to the user, according to her musical taste.

Indeed, in his book [1], Chris Anderson introduces a couple of (among others) very important conditions to exploit the niche markets. These are: *(i)* make everything available, and *(ii)* help me find it. It seems that the former condition is already fulfilled; the distribution and inventory costs are nearly negligible. Yet, to satisfy the latter we need recommender systems that exploit the *"from hits to niches"* paradigm. The main question, though, is whether current recommendation techniques are ready to assist us in this discovery task, providing recommendations of the *hidden jewels* in the Long Tail. In fact, recommenders that appropriately discount popularity may increase total sales [9], as well as potentially increase the margins by suggesting more novel or less known tunes.

Some answers are provided in this paper, in the context of the music domain. The analysis is not performed in terms of classic precision and accuracy of the recommendations, but focusing on how the algorithms behave regarding item popularity. Actually, popularity is the element that defines the characteristic shape of the Long Tail.

This paper is structured as follows: section 2 introduces the Long Tail model. This is the first step needed in order to use artist popularity information. Section 3 focuses on analysing the artists' similarity graph, created using any item–based recommendation algorithm. The metrics allows us to characterise the intrinsic topology of the artist network (e.g. are the hubs in the recommendation network the most popular artists?). Then, section 4 presents the experiments performed in the context of the music domain, comparing three algorithms (collaborative filtering, content–based, and human experts). Finally, in section 5 we discuss the main findings, and conclude with future work in section 6.

2. THE LONG TAIL MODEL

The Long Tail of a catalog is measured in terms of frequency distribution (e.g. purchases, downloads, etc.), ranked by item popularity. It has been largely acknowledged that item popularity can decrease user satisfaction and novelty detection in the recommendation workflow, by providing obvious recommendations [10, 15].

As an example, Figure 1 (left) depicts the Long Tail for 260,525 music artists[1]. The horizontal axis contains the list

[1]The data was gathered from *last.fm* website during July,

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

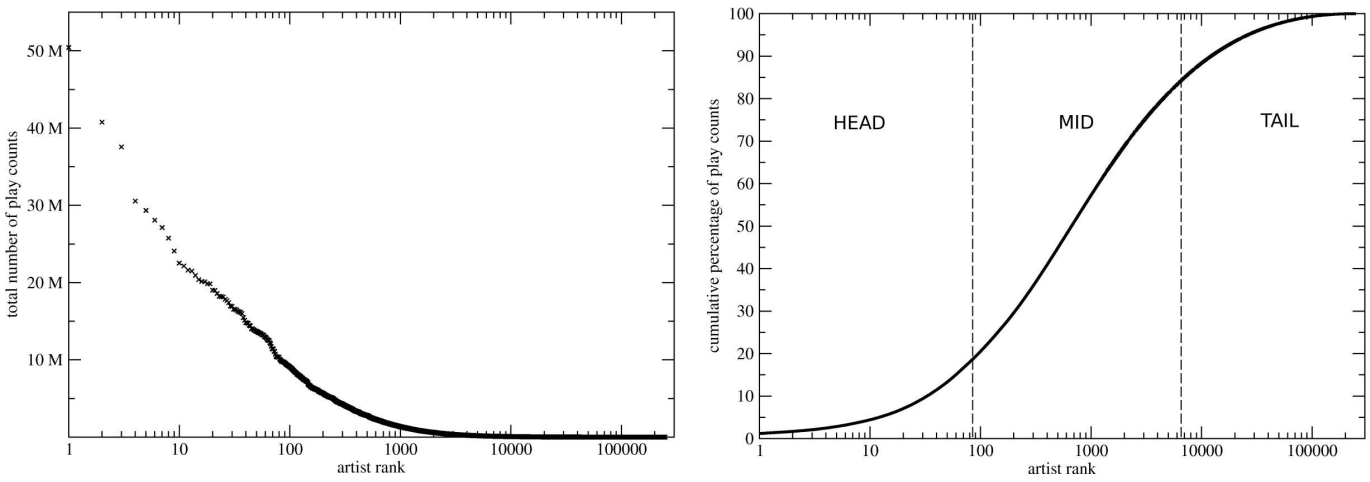

Figure 1: (left) The Long Tail of 260,525 music artists. A log–linear plot depicting artist rank in terms of total playcounts (e.g. at top–1 there is The Beatles with more than 50 million total playcounts). Data gathered from *last.fm*, during July 2007. (right) Cumulative percentage of playcounts from the left figure. Top–737 artists accumulates the 50% of total playcounts (N_{50}). The curve is divided in three parts: head, mid and tail ($X_{head \to mid} = 82$, and $X_{mid \to tail} = 6,655$). The fitted model, $F(x)$, has $\alpha = 0.73$ and $\beta = 1.02$.

of artists ranked by total playcounts. E.g. The Beatles, at position 1, has more than 50 million playcounts.

The Long Tail model, $F(x)$, simulates any Long Tail curve [12]. It models the cumulative distribution of the Long Tail data. $F(x)$ equals to the share of total volume covered by objects up to rank x:

$$F(x) = \frac{\beta}{(\frac{N_{50}}{x})^{\alpha} + 1} \tag{1}$$

where α is the factor that defines the S–shape of the function, β is the total volume share (and also describes the amount of latent demand), and N_{50} is the number of objects that cover half of the total volume, that is $F(N_{50}) = 50$.

Once the Long Tail is modelled using $F(x)$, we can divide the curve in three parts: head, mid, and the tail. The boundary between the head and the mid part of the curve is defined by:

$$X_{head \to mid} = N_{50}^{2/3} \tag{2}$$

Likewise, the boundary between the mid part and the end of the tail is:

$$X_{mid \to tail} = N_{50}^{4/3} \simeq X_{head \to mid}^{2} \tag{3}$$

Figure 1 (right) depicts the cumulative distribution of the Long Tail of 260,525 music artists. Interestingly enough, the top–737 artists account for 50% of the total playcounts, $F(737) = 50$, and only the top–30 artists hold around 10% of the plays. In this sense, the *Gini coefficient* measures the inequality of a given distribution, and it determines the degree of imbalance. In our Long Tail example, 14% of the artists hold 86% of total playcounts, yielding a Gini coefficient of 0.72. This value denotes an imbalanced distribution, higher than the 80/20 Pareto rule (0.6). Figure 1 (right) shows the head of the curve, $X_{head \to mid}$ which consists of only 82 artists, whereas the mid part has 6,573 ($X_{mid \to tail} = 6,655$). The rest of the artists are located in the tail part.

An interesting work is to analyse artist similarity according to the popularity. In our case, this is performed in the context of a network that links the artists (nodes) according to their resemblance. The following section is devoted to explain the metrics that we use.

2007. *Last.fm* provides plugins for virtually any desktop music player to track users' listening behaviour.

3. COMPLEX NETWORK ANALYSIS

We propose several metrics to analyse an item–based recommendation graph; $G := (V, E)$, being V a set of nodes, and E a set of unordered pairs of nodes, named edges. In our case, the items (i.e. music artists) are nodes, and the edges denote the (weighted) similarity among the items, using any item–based recommendation algorithm. The metrics used are derived from Complex Network and Social Network analysis.

3.1 Metrics

3.1.1 Navigation

The **average shortest path** (or mean geodesic length) measures the distance between two vertices i and j. They are connected if one can go from i to j following the edges in the graph. The path from i to j may not be unique. The minimum path distance (or geodesic path) is the shortest path distance from i to j, d_{ij}. The average shortest path in the network is:

$$\langle d \rangle = \frac{1}{\frac{1}{2}n(n+1)} \sum_{i,j \in V, i \neq j} d_{ij} \tag{4}$$

In a random graph, the average path approximates to:

$$\langle d_r \rangle \sim \frac{logN}{log \langle k \rangle}, \tag{5}$$

where $N = |V|$, and $\langle k \rangle$ denotes the mean degree of all the nodes.

The longest path in the network is called its **diameter** (D). In a recommender system, average shortest path and diameter inform us about the global navigation through the network of items.

The **strong giant component**, SGC, of a network is the set of vertices that are connected via one or more geodesics, and are disconnected from all other vertices. Typically, networks possess one large component that contains a majority of the vertices. It is measured as the % of nodes that includes the giant component. In a recommender system, SGC informs us about the catalog coverage, that is the total percentage of available items the recommender recommends to users [10].

3.1.2 Connectivity

The **degree distribution**, p_k, is the number of vertices with degree k:

$$p_k = \sum_{v \in V \mid \deg(v)=k} 1, \tag{6}$$

where v is a vertex, and $\deg(v)$ its degree. More frequently, the *cumulative degree distribution* (the fraction of vertices having degree k or larger), is plotted:

$$P(k) = \sum_{k'=k}^{\infty} p_{k'} \tag{7}$$

A cumulative plot avoids fluctuations at the tail of the distribution and facilitates the evaluation of the power coefficient γ, in case the network follows a power law. In a directed graph, that is when a recommender algorithm only computes the top–n most similar items, $P(k_{in})$ and $P(k_{out})$, the cumulative incoming (outcoming) degree distribution, are more informative. Cumulative degree distribution detects whether a recommendation network has some nodes that act as hubs. That is, that they have a large amount of attached links. This clearly affects the recommendations and navigability of the network.

Another metric used is the **degree correlation**. It is equal to the average nearest–neighbour degree, k^{nn}, as a function of k:

$$k^{\mathrm{nn}}(k) = \sum_{k'=0}^{\infty} k' p(k'|k), \tag{8}$$

where $p(k'|k)$ is the fraction of edges that are attached to a vertex of degree k whose other ends are attached to vertex of degree k'. Thus, $k^{\mathrm{nn}}(k)$ is the mean degree of the vertices we find by following a link emanating from a vertex of degree k.

A closely related concept is the **degree–degree correlation coefficient**, also named *assortative mixing*, which is the Pearson r correlation coefficient for degrees of vertices at either end of a link. A monotonically increasing (decreasing) k^{nn} means that high–degree vertices are connected to other high–degree (low–degree) vertices, resulting in a positive (negative) value of r [17]. In recommender systems, it measures to which extent nodes are connected preferentially to other nodes with similar characteristics.

3.1.3 Clustering

The clustering coefficient, C, estimates the probability that two neighbouring vertices of a given vertex are neighbours themselves. C is defined as the average over the *local measure*, C_i [21]:

$$C_i = \frac{2|E_i|}{k_i(k_i - 1)}, \tag{9}$$

where E_i is the set of existing edges that are direct neighbours of i, and k_i the degree of i. C_i denotes, then, the portion of actual edges of i from the potential number of total edges.

For random graphs, the clustering coefficient is defined as $C_r \sim \langle k \rangle / N$. Typically, real networks have a higher clustering coefficient than C_r.

3.2 Related work in music recommendation

During the last few years, complex network analysis has been applied to music information retrieval in general, and music recommendation in particular. In [6], we compared different music recommendation algorithms based on the network topology. The results are aligned with our main findings: social based recommenders present a scale–free network topology, whereas human expert–based controlled networks does not.

An empirical study of the evolution of a social network constructed under the influence of musical tastes, based on playlist co-occurrence, is presented in [14]. The analysis of collaboration among contemporary musicians, in which two musicians are connected if they have performed in or produced an album together, is presented in [18]. In [2], the authors present a user clustering algorithm that exploits the topology of a user–based similarity network.

A network of similar songs based on timbre similarity is presented in [3]. Interestingly enough, the network is scale–free, thus a few songs appear in virtually any list of similar tracks. This has some problems when generating automatic playlists. [11] presents an analysis of the Myspace social network, and conclude that artists tend to form on-line communities with artists of the same musical genre.

3.3 Network analysis and the Long Tail model

Once each item in the recommendation network is located in the head, mid, or tail part (see section 2), the next step is to combine the similarity network with the Long Tail information. Two main analysis are performed: first, we measure the similarity among the items in each part of the curve. That is, for each item that belongs to the head part, compute the percentage of similar items that are located in the head, mid and tail part (similarly, for the items in the mid and tail part). This measures whether the most popular items are connected with other popular items, and vice versa. Second, we measure the correlation between an item's rank in the Long Tail and its indegree. This allows us to detect whether the hubs in the network are also the most popular items. Section 4 presents the experiments regarding popularity analysis, comparing three different music artists recommendation algorithms: collaborative filtering (CF) from *last.fm*, content–based audio filtering (CB), and expert–based recommendations from *Allmusic.com* (AMG) musicologists.

Property	CF (*Last.fm*)	CB	Expert–based (*AMG*)
N	122,801	59,583	74,494
$\langle k \rangle$	14.13	19.80	5.47
$\langle d_d \rangle$ ($\langle d_r \rangle$)	5.64 (4.42)	4.48 (4.30)	5.92 (6.60)
D	10	7	9
SGC	99.53%	99.97%	95.80%
γ_{in}	2.31($\pm$0.22)	1.61($\pm$0.07)	NA (exp. decay)
r	0.92	0.14	0.17
C (C_r)	0.230 (0.0001)	0.025 (0.0002)	0.027 (0.00007)

Table 1: Artist recommendation network properties for *last.fm* collaborative filtering (CF), content–based audio filtering (CB), and *Allmusic.com* (AMG) expert–based. N is the number of nodes, and $\langle k \rangle$ the mean degree, $\langle d_d \rangle$ is the avg. shortest directed path, and $\langle d_r \rangle$ the equivalent for a random network of size N, and D is the diameter of the network. SGC is the size of the strong giant component, γ_{in} is the power–law exponent of the cumulative indegree distribution, r is the indegree–indegree Pearson correlation coefficient (assortative mixing). C is the clustering coefficient, and C_r for the equivalent random network.

4. EXPERIMENTS

In order to put into practice the Long Tail model, and the properties of item–based recommendation networks, we performed several experiments in the music recommendation field. It is worth noting that music is somewhat different from other entertainment domains, such as movies, or books. Tracking users' preferences are mostly done implicitly, via their listening habits. Moreover, a user can consume an item (i.e. a track, or a playlist) several times, even repeatedly and continuously. Regarding the evaluation process, music recommendation allows us instant feedback with a, say, 30 seconds excerpt.

The experiments aim at evaluating the popularity effect using three (music artists) recommendation approaches: collaborative filtering (CF), content–based audio similarity (CB), and human expert–based resemblance. We measure the popularity effect by contrasting the properties from the network with the Long Tail information of the catalog (e.g. are the hubs in the recommendation network the most popular items? Are the most popular items connected with other popular items, and vice versa?).

4.1 Datasets

CF artist similarity was gathered from *last.fm*, using Audioscrobbler web services[2], and selecting the top–20 similar artists. *Last.fm* has a strong social component, and their recommendations are based on the classic item–based algorithm[3] [20].

To compute artist similarity in the CB network, we apply content–based audio analysis in a music collection ($\mathcal{T}$) of 1.3 Million tracks of 30 seconds samples. Our audio analysis considers not only timbral features (e.g. Mel frequency cepstral coefficients), but some musical descriptors related to rhythm and tonality, among others [7]. Then, to compute artist similarity we used the most representative tracks, $\mathcal{T}_a$, of an artist a, with a maximum of 100 tracks per artist. For each track, $t_i \in \mathcal{T}_a$, we obtain the most similar tracks (excluding those from artist a):

$$sim(t_i) = \underset{\forall t \in \mathcal{T}}{\operatorname{argmin}} \left(distance(t_i, t)\right), \tag{10}$$

and get the artists' names, $\mathcal{A}_{sim(t_i)}$, of the similar tracks. The list of (top–20) similar artists of a is composed by all $\mathcal{A}_{sim(t_i)}$, ranked by frequency and weighted by the audio similarity distance:

$$similar\_artists(a) = \bigcup \mathcal{A}_{sim(t_i)}, \forall t_i \in \mathcal{T}_a \tag{11}$$

Finally, we gather expert recommendations from *All Music Guide* (AMG)[4]. AMG makes use of professional editors to interconnect artists, according to several aspects, such as: *influenced by*, *followers of*, *similar artists*, *performed songs by*, etc. In order to create an homogeneous network, we only make use of the *similar artists* links. Artists from both CB and expert–based networks are a subset of the CF artists.

4.2 Network analysis

The network properties of the three datasets are shown in Table 1. All the networks present the *small–world* phenomena [21]. They have a small average directed shortest path, $\langle d_d \rangle$, similar than its equivalent random network, $\langle d_r \rangle$. Also the clustering coefficients, C, are significantly higher than the equivalent random networks C_r. This is an important property, because recommender systems can be structurally optimised so as to allow users surfing to any part of a music collection with a small number of mouse clicks, and so that they are easy to navigate using only local information [13].

AMG network has a giant component, SGC, smaller than CF and CB networks. Around 4% of their artists are isolated, and cannot be reached from rest (in the giant component). This has strong consequences with regard to the coverage of the recommendations, as well as the navigation for the artists located in the "small islands".

Regarding cumulative indegree distribution, AMG has an exponential decay, whereas CF and CB follow a power law. CF has a power–law exponent, $\gamma = 2.31$, similar to those detected in many scale–free networks, including the world wide web linking structure [5]. These networks are known to show a right–skewed power law distribution, $P(k) \propto k^{-\gamma}$ with $2 < \gamma < 3$, relying on a small subset of hubs that control the network [4].

[2] `http://www.audioscrobbler.net/data/webservices/`

[3] Although, is quite possible that they are using, as well, some information gathered from social tagging.

[4] `http://www.allmusic.com`

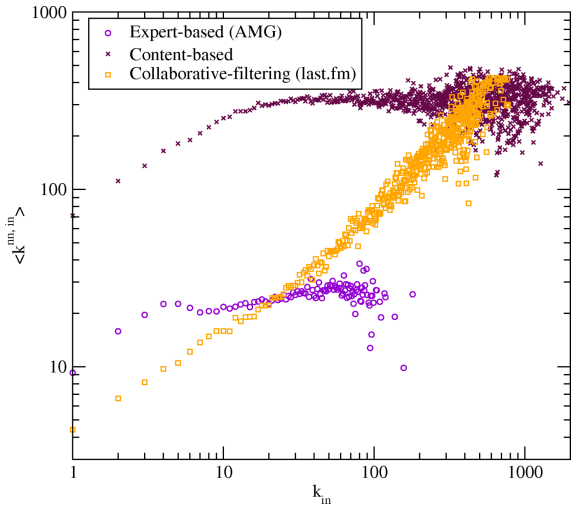

Figure 2: Indegree–indegree correlation (assortative mixing) for the three artist recommendation networks: *last.fm* collaborative filtering (CF), Content–based (CB), and *Allmusic.com* experts. CF clearly presents the assortative mixing phenomenon ($r_{CF} = 0.92$). Neither CB nor Expert–based present any correlation ($r_{CB} = 0.14$, $r_{Expert} = 0.17$).

Another difference is the assortative mixing, or indegree–indegree correlation, presented in Figure 2. CF presents a high assortative mixing ($r = 0.92$). That means that the most connected artists are prone to be similar to other top connected artists. Neither CB nor Expert–based present indegree–indegree correlation, thus artists are connected independently of their inherent properties.

4.3 Popularity analysis

We have outlined, in the previous section, the main topological differences among the three networks. Now, we add the popularity factor (measured in terms of total playcounts per artist), by combining artists' rank in the Long Tail with the results from the network analysis. Two experiments are performed. The former reports the relationships among popular (and unknown) artists. The latter experiment aims at analysing the correlation between artists' indegree and its popularity.

4.3.1 Artist similarity

Figure 3 depicts the correlation among artist's total playcounts and the total playcounts of its similar artists. That is, given the total playcounts of an artist (x axis) it shows, in the vertical axis, the average playcounts of its similar artists. CF network has a clear correlation ($r_{CF} = 0.46$); the higher the playcounts of a given artist, the higher the avg. playcounts of its similar artists. Neither CB nor AMG present any correlation ($r_{CB} = 0.08$, $r_{EX} = 0.09$). Thus, artists are linked independently of their popularity.

Table 2 presents artist similarity divided into the three sections of the Long Tail curve. Given an artist, a_i, it shows (in %) the Long Tail location of its similar artists (results are averaged over all artists). In the CF network, given a very popular artist, the probability of reaching (in one click) a similar artist in the tail is zero. Actually, half of the similar artists are located in the head part, that contains only 82 artists, and the rest in the mid area. Artists in the mid part are tightly related to each other, and only 1/5 of the similar artists are in the tail part. Finally, given an artist in the tail, its similar artists remain in the same area. Contrastingly, CB and expert–based promote much more the mid and tail parts in all the cases (specially in the head part).

Method	$a_i \rightarrow a_j$	Head	Mid	Tail
CF top–20	**Head**	45.32%	54.68%	0%
	Mid	5.43%	71.75%	22.82%
	Tail	0.24%	17.16%	82.60%
CB top–20	**Head**	6.46%	64.74%	28.80%
	Mid	4.16%	59.60%	36.24%
	Tail	2.83%	47.80%	49.37%
Expert	**Head**	5.82%	60.92%	33.26%
	Mid	3.45%	61.63%	34.92%
	Tail	1.62%	44.83%	53.55%

Table 2: Artist similarity and their location in the Long Tail. Given an artist, a_i, it shows (in %) the Long Tail location of its similar artists (results are averaged over all artists). Each row represents, also, the Markov chain transition matrix for CF, CB, and expert–based methods.

Moreover, a Markovian stochastic process [16] is used to simulate someone surfing the recommendation network. Indeed, each row in Table 2 can be seen as a Markov chain transition matrix, M, being the head, mid and tail parts the different states. The values of M denote the transition probabilities, $p_{i,j}$, between two states i, and j (e.g. $p^{CF}_{head,mid} = 0.5468$). The Markovian transition matrix, M^k, denotes the probability of going from any state to another state in k steps (clicks). The initial distribution vector, $P^{(0)}$, sets the probabilities of being at a determined state at the beginning of the process. Then, $P^{(k)} = P^{(0)} \times M^k$, denotes the probability distribution after k clicks, starting in the state defined by $P^{(0)}$.

Using $P^{(k)}$ and defining $P^{(0)} = (1_H, 0_M, 0_T)$, we can get the probability of reaching the tail, starting in the head part. Table 3 shows the number of clicks needed to reach the tail from the head, with a probability $p_{head,tail} \geq 0.4$. In CF, one needs five clicks to reach the tail, whereas in CB and expert–based only two clicks are needed.

Finally, the stationary distribution π is a fixed point (row) vector whose entries sum to 1, and that satisfies $\pi = \pi M$. The last two columns in Table 3 present the stationary distribution vector for each algorithm, and the number of steps to converge to π, with an error $\leq 10^{-6}$. CF needs more than three times the number of steps of CB or expert–based in order to reach the steady state. Even though the probability to stay in the tail in CF is higher than CB and expert–based, this is due to the high probability to remain in the tail once is reached ($p^{CF}_{tail,tail} = 0.8260$).

4.3.2 Artist indegree

Up to now, we have analysed popularity in terms of relationships among the artists. Now we analyse the correlation between artists' indegree (potential hubs in the network) and its popularity. As a starting point, we present in Table 4 the top–10 indegree artists for each network. CF and

Method	k	$\mathbf{P^{(k)}}$, with $P^{(0)} = (1_H, 0_M, 0_T)$ and $p_{head,tail} \geq 0.4$	π	n
CF	5	$(0.075_H, 0.512_M, 0.413_T)$	$(0.044_H, 0.414_M, 0.542_T)$	26
CB	2	$(0.038_H, 0.562_M, 0.400_T)$	$(0.037_H, 0.550_M, 0.413_T)$	7
Expert	2	$(0.030_H, 0.560_M, 0.410_T)$	$(0.027_H, 0.544_M, 0.429_T)$	8

Table 3: Long Tail navigation in terms of a Markovian stochastic process. Second and third columns depict the number of clicks (k) to reach the tail from the head part, with a probability $p_{head,tail} \geq 0.4$. Fourth and fifth columns show the stationary distribution π, as well as the number of steps, n, to reach π.

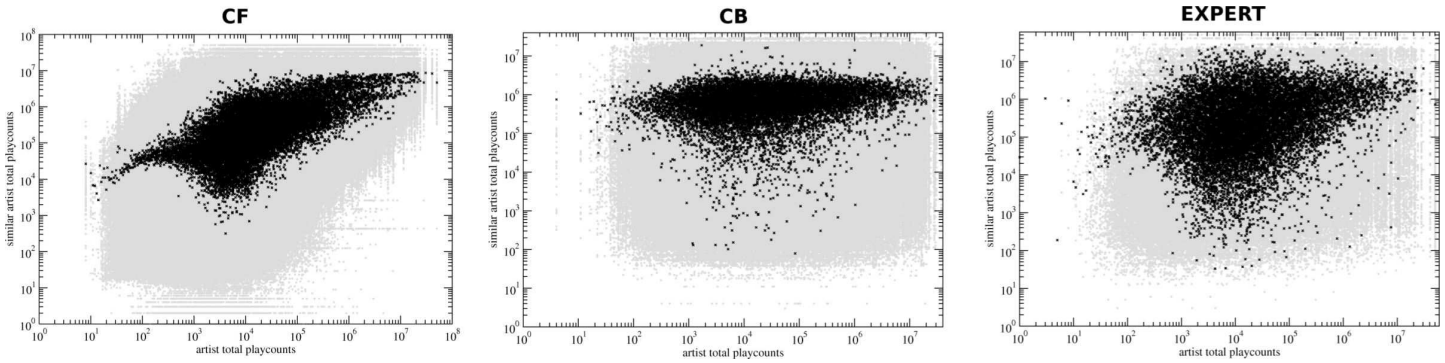

Figure 3: A log–log plot depicting the correlation between artist total playcounts and its similar artists (average values are depicted in black, whilst grey dots display all the values). Pearson r values are: $r_{CF} = 0.46$ $r_{CB} = 0.08$, and $r_{EX} = 0.09$.

CF			CB			Expert		
k_{in}	Artist	LT pos	k_{in}	Artist	LT pos	k_{in}	Artist	LT pos
976	Donald Byrd	6,362	1,955	George Strait	2,632	180	R.E.M.	88
791	Little Milton	19,190	1,820	Neil Diamond	1,974	157	Radiohead	2
772	Rufus Thomas	14,007	1,771	Chris Ledoux	13,803	137	The Beatles	1
755	Mccoy Tyner	7,700	1,646	The Carpenters	1,624	119	David Bowie	62
755	Joe Henderson	8,769	1,547	Cat Stevens	623	117	Nirvana	19
744	R.E.M.	88	1,514	Peter Frampton	4,411	111	Tool	17
738	Wayne Shorter	4,576	1,504	Steely Dan	1,073	111	Pavement	245
717	U2	35	1,495	Lynyrd Skynyrd	668	109	Foo Fighters	45
712	Horace Silver	5,751	1,461	Toby Keith	2,153	104	Soundgarden	385
709	Freddie Hubbard	7,579	1,451	The Charlie Daniels Band	22,201	103	Weezer	51

Table 4: Top–10 artists with higher indegree (k_{in}) for each recommendation network (spikes in Figure 4). The table shows too, the artist ranking in the Long Tail (LT pos).

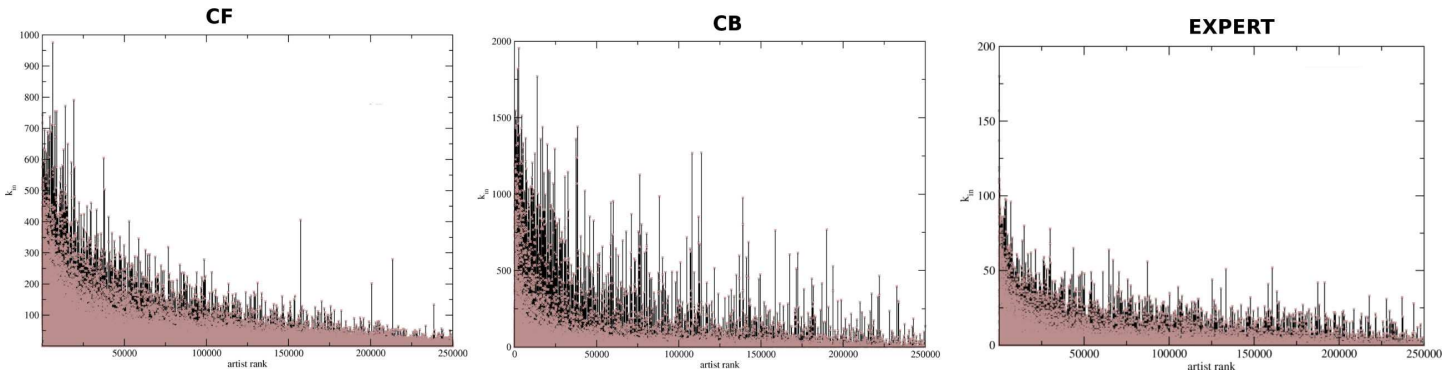

Figure 4: Artist rank in the Long Tail and its indegree, k_{in} (y axis). CF (left) concentrates most of the hubs in the most popular artists—head and mid parts—, whilst in CB (mid), and expert–based (right) hubs are spread out through the whole Long Tail.

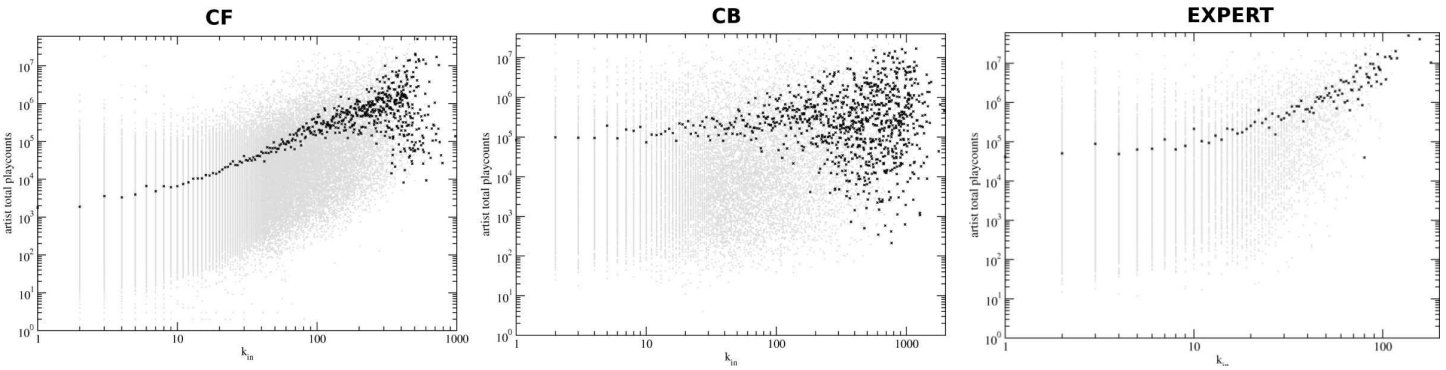

Figure 5: A log–log plot showing the correlation between artist indegree (k_{in}, in horizontal axis) and its total playcounts (avg. values in black), in vertical axis. Pearson r values are: $r_{CF} = 0.38$, $r_{CB} = 0.10$, and $r_{EX} = 0.69$.

expert–based contains two and eight mainstream artists, respectively. CF contains *U2* and *R.E.M.*, but the rest of the list if made of more or less well known Jazz musicians, including some in the—top of the—tail part. The whole list in expert–based AMG is made of very popular artists. Our guess is that the editors connect Long Tail artists with the most popular ones, either for being influential or because a lot of bands are *followers* of these mainstream artists. On the other hand, CB has a more eclectic top–10 list, as one could expect. Oddly enough, there is no new or actual artists, but some classic bands and artists ranging several musical genres. Some bands are, in fact, quite representative of a genre (e.g. *Lynyrd Skynyrd*, and *The Charlie Daniels Band* for Southern Rock, *The Carpenters* for Pop in the 70's, *George Strait* for Country, and *Cat Stevens* for Folk Rock). Probably, the indegree is due to being very influential in its respective musical styles. In some sense, there are some bands that "cite" their music (i.e. sound similar). Although, these results might be somewhat biased; CF and AMG networks are subsets of their whole similar artists' graph, thus our sampling could not be a good representation of the whole dataset. Furthermore, the differences in the maximum indegree value (k_{in} for top–1 artist) among the three networks are due to the different sizes (N) and average degree $\langle k \rangle$, but mostly to the topology of the networks; CF and CB follow a power–law cumulative indegree distribution, whereas AMG has an exponential decay. Therefore, AMG maximum indegree, k_{in}, is much smaller than the CF and CB ones.

Figure 4 depicts the artist rank in the Long Tail and its indegree in the network. The figure shows whether the artists with higher indegree in the network (hubs) are the most popular artists, in terms of total playcounts. In both cases, CF and expert-based networks, the artists with higher indegree (hubs) are mostly located in the head and mid part, whereas in CB they are more spread out through all the curve. In a similar way, Figure 5 presents the correlation between artist indegree (k_{in}), and total playcounts. Again, both CF and AMG expert–based confirm the expectations, as there is a clear correlation between the artist indegree and its total playcounts ($r_{CF} = 0.38$, $r_{EX} = 0.69$). Artists with high indegree are the most popular ones. In CB, given a high indegree value it contains—on average—artists ranging different levels of popularity ($r_{CB} = 0.10$).

5. DISCUSSION

The results show that *last.fm* CF tends to reinforce popular artists, at the expense of discarding less–known music. Thus, the popularity effect derived from the community of users has consequences in the recommendation network. This reveals a somewhat poor discovery ratio when just browsing through the network of similar music artists. It is not easy to reach relevant Long Tail artists, starting from the head or mid parts (see Table 3). Moreover, given a long tail artist, its similar artists are all located in the tail area, too. This do not always guarantee novel music; a user that knows quite well an artist in the Long Tail is likely to know most of the similar artists, too (e.g. the solo project of the band's singer, collaborations with other musicians, and so on). Thus, these might not be considered good novel recommendations to that user, but familiar ones. CF contains, then, all the elements to conclude that popularity has a strong effect in the recommendations because: *(i)* presents assortative mixing (indegree–indegree correlation) in Figure 2, *(ii)* there is a strong correlation between an artist total playcounts and the total playcounts of its similar artists (see Figure 3), *(iii)* most of the hubs in the network are popular artists (see Figure 5), and *(iv)* it is not easy to reach relevant Long Tail artists, starting from the head or mid parts (see Table 3).

Human expert–based recommendations are more expensive to create, and also have a smaller Long Tail coverage compared to automatically generated recommendations like CF and CB. In terms of popularity, the hubs in the expert network are comprised by mainstream music, thus potentially creating a network dominated by popularity (see Table 4 and Figure 5). However, the topology—specially the exponential decay in the indegree distribution—indicates that these artists do not act as hubs. Moreover, it does not present assortative mixing (see Figure 2), so artists are linked in an heterogeneous way; popular artists are connected with other less–known artists (see Table 2 and Figure 3). According to the stationary distribution π (see Table 3), the key Long Tail area in CB and expert–based AMG are the artists located in the mid part. These artists allow to navigate inside the Long Tail acting as entry points, as well as main destinations when leaving the Long Tail. Users that listen to mainly very unknown music are likely to discover artists that are in the mid part, and that are easily reachable

from the artists in the tail. One should pay attention, too, to the quality data in the Long Tail. Assuming that there exists some extremely poor quality music, CB is not able to clearly discriminate against it. In some sense, the popularity effect drastically filters these low quality items. Although, it has been proved in [19] that increasing the strength of social influence increased both inequality and unpredictability of success and, as a consequence, popularity was only partly determined by quality.

Finally, we need to evaluate the quality of the relationships among artists, as well as the popularity effect when providing novel, unknown recommendations to the users. Without any user intervention, then, it is impossible to evaluate the quality and user satisfaction of the recommendations, which does not necessarily correlate with predicted accuracy [15]. In this sense, our incoming work [8] presents a user–centric experiment done with 288 subjects and 5,573 rated songs. The results indicate that even though CF recommends less novel items than CB and expert–based, the users' perceived quality is better than those recommended by CB and human expert methods.

6. CONCLUSIONS

Recommender systems should assist us in the process of filtering and discovering relevant information hidden in the Long Tail. In our experiments, popularity is the element that defines the characteristic shape of the Long Tail. In this sense, we have analysed the popularity effect in three different music recommendation approaches. We measure popularity in terms of total playcounts, and the Long Tail model is used in order to rank all music artists. As expected by its inherent social component, the collaborative filtering approach is prone to popularity bias. This has some consequences on the discovery ratio as well as in the navigation through the Long Tail.

Future work includes expanding the analysis of the recommendation network, taking into account its dynamics. This could be used, for instance, to detect "hype" items, that become popular in a very short period of time.

7. ACKNOWLEDGMENTS

This work was partially funded by the Pharos (IST-FP6-45035) and Variazoni (e-Content plus) projects, sponsored by the European Commission.

8. REFERENCES

[1] C. Anderson. *The Long Tail. Why the future of business is selling less of more*. Hyperion Verlag, 2006.

[2] A. Anglade, M. Tiemann, and F. Vignoli. Complex-network theoretic clustering for identifying groups of similar listeners in p2p systems. In *Proceedings of the ACM conference on Recommender systems*, pages 41–48, Minneapolis, USA, 2007. ACM.

[3] J.-J. Aucouturier and F. Pachet. A scale-free distribution of false positives for a large class of audio similarity measures. *Pattern Recognition*, 41(1):272–284, 2008.

[4] A. L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, October 1999.

[5] A.-L. Barabási, R. Albert, H. Jeong, and G. Bianconi. Power-law distribution of the world wide web. *Science*, 287:2115a, 2000.

[6] P. Cano, Ò. Celma, M. Koppenberger, and J. Martin-Buldú. Topology of music recommendation networks. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 16(013107), 2006.

[7] P. Cano, M. Koppenberger, and N. Wack. An industrial-strength content-based music recommendation system. In *Proceedings of 28th International ACM SIGIR Conference*, Salvador, Brazil, 2005.

[8] Ò. Celma and P. Herrera. Evaluating the quality of novel recommendations. In *Proceedings of the 2nd ACM International Conference on Recommender Systems*, Laussane, Switzerland, 2008.

[9] D. M. Fleder and K. Hosanagar. Blockbuster culture's next rise or fall: The impact of recommender systems on sales diversity. *SSRN eLibrary*, 2007.

[10] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, 2004.

[11] K. Jacobson and M. Sandler. Musically meaningful or just noise? an analysis of on-line artist networks. In *Proceedings of the 6th International Symposium on Computer Music Modeling and Retrieval*, Copenhagen, Denmark, 2008.

[12] K. Kilkki. A practical model for analyzing long tails. *First Monday*, 12(5), May 2007.

[13] J. M. Kleinberg. Navigation in a small world. *Nature*, 406:845, 2000.

[14] J. Martin-Buldú, P. Cano, M. Koppenberger, J. Almendral, and S. Boccaletti. The complex network of musical tastes. *New Journal of Physics*, 9(172), 2007.

[15] S. M. Mcnee, J. Riedl, and J. A. Konstan. Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *Computer Human Interaction*, pages 1097–1101, New York, USA, 2006. ACM.

[16] S. P. Meyn and R. L. Tweedie. *Markov chains and stochastic stability*. Springer–Verlag, 1993.

[17] M. E. J. Newman. Assortative mixing in networks. *Physical Review Letters*, 89(20), 2002.

[18] J. Park, Ò. Celma, M. Koppenberger, P. Cano, and J. Martin-Buldú. The social network of contemporary popular musicians. *International Journal of Bifurcation and Chaos*, 17(7):2281–2288, 2007.

[19] M. J. Salganik, P. S. Dodds, and D. J. Watts. Experimental study of inequality and unpredictability in an artificial cultural market. *Science*, 311(5762):854–856, February 2006.

[20] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In A. Press, editor, *Proceedings of 10th International World Wide Web Conference*, pages 285–295, Hong Kong, 2001.

[21] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, June 1998.

Investigation of Various Matrix Factorization Methods for Large Recommender Systems

Gábor Takács*
Dept. of Mathematics and Computer Science
István Széchenyi University
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

Matrix Factorization (MF) based approaches have proven to be efficient for rating-based recommendation systems. In this work, we propose several matrix factorization approaches with improved prediction accuracy. We introduce a novel and fast (semi)-positive MF approach that approximates the features by using positive values for either users or items. We describe a momentum-based MF approach. A transductive version of MF is also introduced, which uses information from test instances (namely the ratings users have given for certain items) to improve prediction accuracy. We describe an incremental variant of MF that efficiently handles new users/ratings, which is crucial in a real-life recommender system. A hybrid MF–neighbor-based method is also discussed that further improves the performance of MF. The proposed methods are evaluated on the Netflix Prize dataset, and we show that they can achieve very favorable Quiz RMSE (best single method: 0.8904, combination: 0.8841) and running time.

Categories and Subject Descriptors

H.5.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Information Filtering*; H.2.8 [**Database Management**]: Database Applications—*Data Mining*

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

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

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

General Terms

Algorithms, Experimentation

Keywords

recommender systems, collaborative filtering, Netflix Prize, matrix factorization, neighbor-based methods, incremental gradient descent methods

1. INTRODUCTION

Recommender systems attempt to profile user preferences over items and models the relation between users and items. The task of recommender systems is to recommend items that fit the user's taste, in order to help the user in selecting/purchasing items from an overwhelming set of choices. Such systems have great importance in applications such as e-commerce, subscription-based services, information filtering, etc. Recommender systems providing personalized suggestions greatly increase the likelihood of a customer making a purchase compared to unpersonalized ones. Personalized recommendations are especially important in markets where the variety of choices is large, the taste of the customer is important, and typically 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.

With the growing significance of e-commerce, an increasing number of web-based merchant and rental services use recommender systems. Some of the major participants of e-commerce web, like Amazon.com and Netflix, successfully apply recommender systems to deliver automatically generated personalized recommendation to their customers. The importance of a good recommender system was recognized by Netflix, which led to the announcement of the Netflix Prize (NP) competition to motivate researchers to improve the accuracy of their recommender system called Cinematch. This competition motivated our present work as well.

The approach which makes use of only user activities of the past[1] (e.g. transaction history or user satisfaction expressed in rating) is termed *collaborative filtering* (CF). The

[1]In contrast to the *content-based approaches* which use also demographic data to profile users.

NP contest focuses on the case when users express their opinion of items by means of ratings. In this framework, the user first provides ratings of some items usually on a discrete numerical scale, and the system then recommends other items based on ratings similar users have already provided.

Matrix factorization based techniques have proven to be efficient in recommender systems (see Section 1.1) when predicting user preferences from known user-item ratings. The main contribution of this work is a number of novel MF based algorithms that are accurate in predicting user ratings, and provide scalable solutions for large-scale recommender systems. In particular, we present

- an MF with biases, which is currently our best performing approach.
- a novel and fast (semi-)positive MF approach that approximates the factors by using positive values for either users or items;
- a momentum-based MF approach;
- a transductive version of MF that makes use of information from test instances (namely the ratings users have given for certain items) to improve prediction accuracy;
- an incremental variant of MF that efficiently handles new users/ratings (this is crucial in a real-life recommender systems);
- a hybrid MF–neighbor based method is introduced, which improves the accuracy of MF considerably.

The proposed methods were evaluated on the Netflix Prize problem, but this does not limit their applicability to this specific dataset. The presented methods are parts of the blended solution of our team Gravity in the NP contest.

1.1 Related Work

The first works on the field of CF have been published in the early 1990s. The Tapestry system [7] used collaborative filtering to filter mails simultaneously from several mailing lists based on the opinion of the community on readings. Over the last broad decade many CF algorithms have been proposed that approach the problem by different techniques, including similarity/neighborhood based approaches [10, 13], Bayesian networks [6], restricted Boltzman machines (RBM) [12], and various matrix factorization techniques [8, 14].

The NP competition boosted the interest in CF, and yielded a number of related publications. We should here mention the NP related 1st Netflix-KDD Workshop in 2007 [4], which brought together top contenders of the contest. The members of BellKor/KorBell team[2] presented an improved neighborhood based approach in [2], which removes the global effect from the data—can be considered as normalization—to improve the accuracy of similarity based interpolative predictions. Paterek applied successfully various matrix factorization techniques [9] by adding biases to the regularized MF, postprocessing the residual of MF with kernel ridge regression, using a separate linear model for each movie, and by decreasing the parameters in regularized MFs.

Our methods are different from the above ones in various aspects. BellKor uses alternate least squares, but we use incremental gradient descent method at weight updates. Their method does not use the chronological order of ratings, while we exploit this information in our approaches. The accuracy of their published MF variants is inferior to ours. They use only positive and normal MFs, while we propose the semi-positive version of MF algorithm. The learning of BellKor's positive MF is more complicated and consequently significantly slower than ours, but this complexity does not yield improvement on the accuracy. Paterek apply a different learning scheme that compares unfavorable to ours in terms of speed and accuracy. We point out that this difference result in faster training and better accuracy of our MF methods. Other differences are that Paterek uses less meta-parameters for his MF methods. The idea of using test data has also appeared at various authors for different approaches: RBM in [12], LM, NSVD1 and NSVD2 in [9]. Our presented approaches differ slightly from known ones but these modifications are together important: simultaneous feature training, regularization, bias features, early stopping criteria, incremental gradient descent training algorithm, and date based ordering of the ratings of users.

[2]Winner of the Progress Prize 2007, awarded to the leading team at the one-year anniversary of NP.

2. PROBLEM DEFINITION

We define the problem of collaborative filtering in the following setting. A set of I users and a set of J items are given. A rating record is a quadruple (i, j, d_{ij}, x_{ij}) representing that user i rated item j on date d_{ij} as x_{ij}, where $i \in \{1, \dots, I\}$, $j \in \{1, \dots, J\}$, $d_{ij} \in \mathcal{D}$ the ordered set of possible dates, and $x_{ij} \in \mathcal{X} \subset \mathbb{R}$. Typical rating values can be binary ($\mathcal{X} = \{0, 1\}$), integers from a given range (e.g. $\mathcal{X} = \{1, 2, 3, 4, 5\}$), or real numbers of a closed interval (e.g. $\mathcal{X} = [-1, 10]$). We assume that a given user can rate a given item at most once. This justifies the use of subscripts ij for dates and rating values. We are given a finite set of rating records, $\mathcal{T}$, which are used for the training. We refer to the set of all known (i, j) pairs in $\mathcal{T}$ as $\mathcal{R}$. Note that typically $|\mathcal{R}| \ll |I| \cdot |J|$, because each user rates only a few items.

The rating values can be organized in a rating matrix $\mathbf{X}$ where elements indexed by $(i, j) \notin \mathcal{R}$ are unknown. In this paper we adopt the evaluation measure of NP contest, the root mean squared error (RMSE), which is defined as:

$$RMSE(T) = \sqrt{\frac{1}{|T|} \sum_{(i,j) \in T} (\hat{x}_{ij} - x_{ij})^2}, \qquad (1)$$

where T (i, j) contains user-item pairs on which the ratings are predicted. The accuracy of predictors are evaluated on a validation set $\mathcal{V}$; naturally the ratings of $\mathcal{V}$ is not used at the creation of the predictor. Since in recommender systems the goal is to predict the user preferences from past ratings, the ratings of $\mathcal{T}$ precedes the ratings of $\mathcal{V}$ in time.

From now on, without loss of generality, we use terms item and movie as synonyms. 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.

3. MATRIX FACTORIZATION METHODS

This section exploits our previous work [15] to introduce matrix factorization. We there discussed Basic MF and Regularized MF methods, and the incorporation of constant values in the matrices. The goal of MF techniques is to approximate $\mathbf{X}$ as a product of two much smaller matrices:

$$\mathbf{X} \approx \mathbf{UM}, \qquad (2)$$

where $\mathbf{U}$ is an $I \times K$ and $\mathbf{M}$ is a $K \times J$ matrix.

3.1 Basic MF

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 $\mathbf{U} \in \mathbb{R}^{I \times K}$ and $\mathbf{M} \in \mathbb{R}^{K \times J}$. Let u_{ik} denote the elements of $\mathbf{U}$, and m_{kj} the elements of $\mathbf{M}$. Let $\mathbf{u}_i^T$ denote a row of $\mathbf{U}$, and $\mathbf{m}_j$ a column of $\mathbf{M}$. Then:

$$\hat{x}_{ij} = \sum_{k=1}^{K} u_{ik} m_{kj} = \mathbf{u}_i^T \mathbf{m}_j \tag{3}$$

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

$$e'_{ij} = \frac{1}{2} e_{ij}^2 \tag{4}$$

$$\text{SSE} = \sum_{(i,j)\in\mathcal{R}} e_{ij}^2, \qquad \text{SSE}' = \frac{1}{2}\text{SSE} = \sum_{(i,j)\in\mathcal{R}} e'_{ij}$$

$$\text{RMSE} = \sqrt{\text{SSE}/|\mathcal{R}|} \tag{5}$$

$$(\mathbf{U}^*, \mathbf{M}^*) = \arg\min_{(\mathbf{U},\mathbf{M})} \text{SSE}' = \arg\min_{(\mathbf{U},\mathbf{M})} \text{SSE} = \arg\min_{(\mathbf{U},\mathbf{M})} \text{RMSE} \tag{6}$$

Here $\hat{x}_{ij}$ denotes how the i-th user would rate the j-th movie, according to the model, e_{ij} denotes the training error on the (i,j)-th rating, and SSE denotes the sum of squared training errors. Eq. (6) 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 RMSE, which is equivalent to minimize SSE$'$, we have applied a simple incremental gradient descent method to find a local minimum of SSE$'$, where one gradient step intend to decrease the square of prediction error of only one rating, or equivalently, either e'_{ij} or e_{ij}^2. Suppose we are at the (i,j)-th training example, x_{ij} and its approximation $\hat{x}_{ij}$ is given.

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

$$\frac{\partial}{\partial u_{ik}} e'_{ij} = -e_{ij} \cdot m_{kj}, \quad \frac{\partial}{\partial m_{kj}} e'_{ij} = -e_{ij} \cdot u_{ik}. \tag{7}$$

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

$$u'_{ik} = u_{ik} + \eta \cdot e_{ij} \cdot m_{kj}, \quad m'_{kj} = m_{kj} + \eta \cdot e_{ij} \cdot u_{ik}. \tag{8}$$

that is, we change the weights in $\mathbf{U}$ and $\mathbf{M}$ to decrease the square of actual error, thus better approximating x_{ij}. Here η is the learning rate. We refer to this method as Basic MF.

3.2 Regularized MF

Consider the following case: let $K = 2$, $m_{71} = 1$, $m_{72} = 0$, $m_{81} = 1$, $m_{82} = 0.1$, $x_{67} = 4$, $x_{68} = 3$, $|\{j : (i,j) \in \mathcal{R}, i = 6\}| = 2$. In other words, we have two features, and the 6th user rated only two very similar movies, the 7th and the 8th, as a 4 and a 3. In this case, according to eq. (6), the optimal user features are $u_{61} = 4$, $u_{62} = -10$, which perfectly describe the ratings of this user: $\hat{x}_{67} = 4, \hat{x}_{68} = 3$.

Apparently such large feature values ($x_{62} = -10$) should be avoided. The common way of overcoming this consists in applying regularization by penalizing the square of the Euclidean norm of weights, which results in a new optimization problem:

$$e'_{ij} = (e_{ij}^2 + \lambda \cdot \mathbf{u}_i^T \cdot \mathbf{u}_i + \lambda \cdot \mathbf{m}_j^T \cdot \mathbf{m}_j)/2 \tag{9}$$

$$\text{SSE}' = \sum_{(i,j)\in\mathcal{R}} e'_{ij} \tag{10}$$

$$(\mathbf{U}^*, \mathbf{M}^*) = \arg\min_{(\mathbf{U},\mathbf{M})} \text{SSE}' \tag{11}$$

Note that minimizing SSE$'$ is no longer equivalent to minimizing SSE. Similar to the Basic MF approach, we compute the gradient of e'_{ij}, and update the weights in the direction opposite of the gradient:

$$\frac{\partial}{\partial u_{ik}} e'_{ij} = -e_{ij} \cdot m_{kj} + \lambda \cdot u_{ik}, \quad \frac{\partial}{\partial m_{kj}} e'_{ij} = -e_{ij} \cdot u_{ik} + \lambda \cdot m_{kj}. \tag{12}$$

$$u'_{ik} = u_{ik} + \eta \cdot (e_{ij} \cdot m_{kj} - \lambda \cdot u_{ik}), \tag{13}$$

$$m'_{kj} = m_{kj} + \eta \cdot (e_{ij} \cdot u_{ik} - \lambda \cdot m_{kj}) \tag{14}$$

For the learning algorithm used in Regularized MF, see Algorithm 1.

Input: $\mathcal{R}, \mathbf{X}, \eta, \lambda, \mathrm{T}, \mathcal{V}$
Output: $\mathbf{U}^*, \mathbf{M}^*$
Initialize $\mathbf{U}$ and $\mathbf{M}$ with small random numbers.
loop until the terminal condition is met. One epoch:
  iterate over each (i,j) element of T. For x_{ij}:
    compute e'_{ij};
    compute the gradient of e'_{ij}, according to (12);
    for each $k \in \{1, \ldots, K\}$
      update the i-th row of $\mathbf{U}$ and the j-th column of $\mathbf{M}$ according to (13)–(14);
  calculate the RMSE on $\mathcal{V}$;
  if the RMSE on $\mathcal{V}$ was better than in any prev. epoch:
    Let $\mathbf{U}^* = \mathbf{U}$ and $\mathbf{M}^* = \mathbf{M}$.
  terminal condition: RMSE on $\mathcal{V}$ does not decrease during two epochs.
end

Algorithm 1: Training algorithm for Regularized MF

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

3.3 BRISMF

We found a simple way to boost the performance of Regularized MF, by fixing the first column of $\mathbf{U}$ and the second row of $\mathbf{M}$ to the constant value of 1. Under the expression "fixing to a constant value" we mean not to apply eqs. (13)–(14) when updating u_{i1} and m_{2j}, and in the initialization, to assign them the constant value instead of random values. The pair for these features (m_{1j} and u_{i2}) can serve as a bias feature. This simple extension speeds up the training phase and yields a more accurate model with better generalization performance. We refer to this method as BRISMF that stands for Biased Regularized Incremental Simultaneous MF. In [15] the insertion of constant values into $\mathbf{U}$ and $\mathbf{M}$ was proposed. BRISMF is a special way of inserting constant values: all the inserted values are 1-s. We remark that the bias feature idea was mentioned also by Paterek in [9],

3.4 Semipositive and positive MF

The presented Regularized MF algorithm can assign not only positive but also negative feature values. Positive matrix factorization techniques have been succesfully applied in the field of CF [8]. We present a simple extension of Regularized MF that can give positive and semipositive factorizations. We talk about semipositive MF when exactly one of $\mathbf{U}$ and $\mathbf{M}$ contains both positive and negative values, and positive MF, when both contains only non-negative values.

The idea can be summarized as follows: for the (i,j)-th training example in a given epoch, if u_{ik} or m_{kj} becomes negative due to the application of eqs. (13)–(14), we reset it to 0. We describe the modified equations for the case when both user and movie features are required to be non-negative:

$$u'_{ik} = \max\{0, u_{ik} + \eta \cdot e_{ij} \cdot m_{kj} - \lambda \cdot u_{ik}\} \quad (15)$$

$$m'_{kj} = \max\{0, m_{kj} + \eta \cdot e_{ij} \cdot u_{ik} - \lambda \cdot m_{kj}\} \quad (16)$$

If we allow user features to be negative, we can simply use eq. (13) instead of eq. (15). Allowing only negative movie features can be treated similarly.

The presented approach can easily be extended to handle arbitrary box-constraints (i.e. prescribing a minimum and a maximum possible value for features e.g. 0 and 1). The handling of arbitrary box constraints can be thought of as a generalization of constant features, for example in the case of bias features, $1 \le u_{i1} \le 1$ and $1 \le m_{2j} \le 1$.

Though the (semi-)positive MF has no advantage over Regularized MF in the NP Problem, in real recommender systems the model can provide a better explanation of users' behaviour. When box constraints are 0 and 1, the feature values can be thought of as fuzzy membership values.

3.5 Applying momentum method

We can apply momentum method by a small modification of the original learning rule. In each learning step the modification of the weights are calculated not only from the actual gradient but from the last weight changes too. With the modification of (13) and (14) we get the following equations:

$$u_{ik}(t+1) = u_{ik}(t) + \Delta u_{ik}(t), \quad (17)$$

$$\Delta u_{ik}(t) = \eta \cdot (e_{ij} \cdot m_{kj} - \lambda \cdot u_{ik}(t)) + \sigma \cdot \Delta u_{ik}(t-1),$$

$$m_{kj}(t+1) = m_{kj}(t) + \Delta m_{kj}(t), \quad (18)$$

$$\Delta m_{kj}(t) = \eta \cdot (e_{ij} \cdot u_{ik} - \lambda \cdot m_{kj}(t)) + \sigma \cdot \Delta m_{kj}(t-1)$$

Here σ is the momentum factor. Though momentum MF is not among our best MFs, it blends well with other MF methods.

3.6 Retraining user features

The Regularized MF algorithm has a serious disadvantage: movie features change while we iterate through users. If the change is large, user features updated in the beginning of an epoch may be inappropriate at the end of the epoch.

To overcome this problem, we can completely recompute the user features after the learning procedure. This method can serve as an efficient incremental training method for recommender systems. The algorithm can be summarized as follows:

1. First training: apply an MF algorithm.
2. Let $\mathbf{M} = \mathbf{M}^*$. Initialize $\mathbf{U}$ randomly.
3. Second training: Apply the same learning procedure as in the first training step, with the following restrictions: skip the weight initialization step; do not change weights in $\mathbf{M}$, i.e. do not apply eq. (14) or its variants; store the optimal number of epochs, denote it n^*.

Note that this method can efficiently handle the addition of new users, or new ratings for an existing user, which is very important in a real recommender system: we do not apply the whole training procedure, just reset the weights of a certain user, and apply the second training procedure for n^* epochs, only for that certain user. Note also, that the second training procedure needs to iterate through the entire database, which requires slow disk access operation, only once (not n^* times), as the ratings of a user can be kept in memory and can be immediately re-used in the next epoch.

We found a simple modification of this algorithm very effective: in the second training, learn not only $\mathbf{U}$, but $\mathbf{M}$ as well. Though this is not useful for incremental learning in recommender systems anymore, it boosts the performance of MF.

3.7 Transductive MF

Transductive learning involves the use of test examples but not their labels. In the case of CF, this means that the algorithm "knows" what test examples the model will be applied for, i.e. the $(i,j) \in \mathcal{V}$ pairs, but not the corresponding x_{ij} values. For Restricted Boltzmann Machines there is a straightforward way to get a better model by incorporating the test set [12].

We have developed a Transductive MF that can incorporate $(i,j) \in \mathcal{V}$ information after the learning process. The idea behind the method is the following: suppose that user i has the following n ratings in the test set (test ratings) to be predicted: $x_{i1}, \ldots, x_{in}$. The user has a feature vector ($\mathbf{u_i}$). When we are at the j-th example, we can predict another feature vector based only on what other movies ($1 \le j' \le n, j' \ne j$) are to be predicted. A proper linear combination – not depending on i or j – of the original user feature vector and this new feature vector can yield another feature vector for the prediction of x_{ij} that is better than the original one. Formally:

$$\mathbf{u}'_i(j) = \frac{1}{\sqrt{|j' : (i,j') \in \mathcal{R}| + 1}} \sum_{\substack{j''=1 \\ j'' \ne j}}^{n} \mathbf{m}_{j''}. \quad (19)$$

$$\hat{x}'_{ij} = \hat{x}_{ij} + \nu \cdot \mathbf{u}'_i(j)^T \cdot \mathbf{m}_j = \mathbf{u}_i^T \cdot \mathbf{m}_j + \nu \cdot \mathbf{u}'_i(j)^T \cdot \mathbf{m}_j.$$

The attenuation factor in eq. (19) ensures that the more ratings a user has in a training set, the less information the test set provides, thus $\hat{x}'_{ij}$ will differ less from $\hat{x}_{ij}$. In practice, ν need not be determined: we can use $\hat{x}'_{ij}$ and $\mathbf{u}'_i(j)^T \cdot \mathbf{m}_j$ as two predictions for x_{ij}, and apply linear regression to get the best RMSE.

Though transductive methods cannot be applied directly in real applications, the transductive MF can boost up the performance of a predictor for the Netflix Prize problem.

3.8 Neighbor based correction of matrix factorization

It is known that the combination of MF and NB can lead to very accurate predictions [2, 3]. However, conventional NB methods scale up poorly for large problems. The price

of additional accuracy is the loss of scalability.

Here we propose a scalable scheme for using the MF and NB approaches together. The idea is that we improve an existing MF model ($\mathbf{U}$,$\mathbf{M}$) by adding an item neighbor based correction term to its answer in the prediction phase. The corrected answer for query (i, j) is the following:

$$\hat{r}_{ij} = \mathbf{u}_i^T \mathbf{m}_j + \gamma \frac{\sum_{k \in \mathcal{T}_i \setminus \{j\}} s_{jk} \left(\mathbf{u}_i^T \mathbf{m}_k - r_{ik} \right)}{\sum_{k \in \mathcal{T}_i \setminus \{j\}} s_{jk}},$$

where s_{jk} is the similarity between items j and k, and $\mathcal{T}_i$ is the set items rated by user i. The weight of the correction term γ can be optimized via cross-validation.

The similarity s_{jk} can be defined in many different ways. Here are two variants that proved to be useful for the Netflix Prize problem [5].

- (S1): Normalized scalar product based similarity.

$$s_{jk} = \left(\frac{\sum_{\ell=1}^{K} m_{\ell j} m_{\ell k}}{\sqrt{\sum_{\ell=1}^{K} m_{\ell j}^2} \cdot \sqrt{\sum_{\ell=1}^{K} m_{\ell k}^2}} \right)^{\alpha},$$

  where α is the amplification parameter.

- (S2): Normalized Euclidean distance based similarity.

$$s_{jk} = \left(\frac{\sum_{\ell=1}^{K} (m_{\ell j} - m_{\ell k})^2}{\sqrt{\sum_{\ell=1}^{K} m_{\ell j}^2} \cdot \sqrt{\sum_{\ell=1}^{K} m_{\ell k}^2}} \right)^{\alpha}.$$

In both cases, the value s_{jk} can be calculated in $\mathrm{O}(K)$ time, thus $\hat{r}_{ij}$ can be calculated in $\mathrm{O}(K \cdot |\mathcal{T}_i|)$. We remark that one can restrict to use only the top S neighbors of the queried item [2], however, it does not affect the time requirement, if we use the same function for s_{ij} and neighbor-selection.

Now let us comment in more details on the time and memory requirement of our NB corrected MF in comparison with the improved neighborhood based approach of BellKor [2], which can also be applied to further improve the results of an MF. For a give query, the time requirement of our method is $\mathrm{O}(K \cdot S)$, while their method requires to solve a separate linear least squares problem with S variables, thus it is $\mathrm{O}(S^3)$. Memory requirements: for the i-th user, our method requires to store in the memory $\mathbf{u}_i$ and $\mathbf{M}$, that is $\mathrm{O}(KJ)$, while their approach necessitates to store the item-by-item matrix and the ratings of the user, which is $\mathrm{O}(J^2 + |\mathcal{T}_i|)$. For all users, our method requires $\mathrm{O}(IK + KJ)$ while their approach requires $\mathrm{O}(J^2 + |\mathcal{R}|)$ memory.

Regardless of the simplicity of our method its effectiveness is comparable with that of Bell and Koren's method, see Section 4.

This model can be seen as a nice unification of the MF and NB approaches. It is simple, scalable, and accurate. The training is identical to the regular MF training. The prediction consists of an MF and a NB term. The similarities used in the NB term need not to be precomputed and stored, because they can be calculated very efficiently from the MF model.

4. EXPERIMENTAL STUDY

The experimentations have been performed on the Netflix Prize dataset, being currently the largest available one for the public. It contains about 100 million ratings from over 480k users on nearly 18k movies. For more information on the dataset see e.g. [3, 4]. We decided to validate our algorithm against the Netflix dataset since currently this is the most challenging problem for the CF community due to its challenging properties (see e.g. [1] for a comprehensive summary of these properties), and the task of the Netflix Grand Prize.

In this experimentation section we evaluate the presented methods on a randomly selected 10% subset of the Probe set[3], which we refer to as Probe10. [4] We mention that we measured only a slight difference between the RMSE values on the two test sets: our Probe10 and Quiz[5]. For some selected methods, we also report on Quiz RMSE values. We performed all tests on a 2 GHz Intel Pentium M (Dothan) laptop with 1 GB RAM.

4.1 Parameter settings

All of the presented methods have many pre-defined parameters that greatly influences the result. A model and the corresponding parameter setting can be considered useful if either it results in a very accurate model (low test RMSE) or the model "blends well" i.e. improves the accuracy of the blended solution. For combining different models we applied ridge regression.[6]

We applied a very simple but effective parameter optimization algorithm. We initialize parameters randomly, then we select a single parameter to optimize. We assign n (typically 2) distinct random values to the parameter, and choose the best performing one by evaluating MF with this parameter setting. This method is performed systematically for all parameters one-by-one.

4.2 Results of Matrix Factorization

We ordered the training examples user-wise and then by date. By this we model that user's taste change in time, and the most recent taste is the dominant. This coincides with the creation of train-test split of NP dataset. We performed test runs with numerous parameters settings. We report on the actual settings of the parameters at each result. Naming convention for parameters: learning rate and regularization factor for users and movies $(\eta_u, \eta_m, \lambda_u, \lambda_m)$; the corresponding variables of bias features $(\eta_{ub}, \eta_{mb}, \lambda_{ub}, \lambda_{mb})$; minimum and maximum weights in the uniform random initialization of $\mathbf{U}$ and $\mathbf{M}$: $w_{\underline{u}}, w_{\overline{u}}, w_{\underline{m}}, w_{\overline{m}}$; offset G to subtract from $\mathbf{X}$ before learning.

4.2.1 Comparing Regularized MF and BRISMF

We compare two MFs:

- a Regularized MF, which we name briefly as RegMF#0, with the following parameter settings: $K = 40, \eta = 0.01, \lambda = 0.01, w_{\underline{u}} = -w_{\overline{u}} = w_{\underline{m}} = -w_{\overline{m}} = -0.01$
- a BRISMF, which we name briefly as BRISMF#0, with the following parameter settings: $K = 40, \eta = 0.01, \lambda = 0.01, w_{\underline{u}} = -w_{\overline{u}} = w_{\underline{m}} = -w_{\overline{m}} = -0.01$

[3]Probe set: dedicated for testing purpose by Netflix with known ratings

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

[5]Quiz set: to be predicted data, but only Netflix knows the ratings.

[6]The source code of our combination algorithm is publicly available at our web site.

RegMF#0 reaches its optimal Probe10 RMSE in the 13th epoch: Probe10 RMSE is 0.9214, while these numbers for BRISMF#0 are: 10th and 0.9113, which is a 0.0101 improvement.

4.2.2 Changing the parameters of the BRISMF

Table 1 present the influence of η and λ on the Probe10 RMSE. Other parameters are the same as in BRISMF#0. Best result: $\eta = 0.007, \lambda = 0.005$. We refer to this MF in the following as BRISMF#1. The running time for this MF is only 14 minutes! The number of epochs to get this RMSE was 10. Note that running time depends only on K and the number of epochs to get the optimal Probe10 RMSE.

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

η \ λ	0.005	0.007	0.010	0.015	0.020
0.005	0.9061	0.9079	0.9117	0.9168	0.9168
0.007	**0.9056**	0.9074	0.9112	0.9168	0.9169
0.010	0.9064	0.9077	0.9113	0.9174	0.9186
0.015	0.9099	0.9111	0.9152	0.9257	0.9390
0.020	0.9166	0.9175	0.9217	0.9314	0.9431

4.2.3 Subsampling users

On Figure 1 we demonstrate how the number of users (thus, the number of ratings) influences Probe10 RMSE and the optimal number of training epochs in case of BRISMF#1. Probe10 RMSE varies between 0.9056 and 0.9677, and the number of epochs between 10 and 26. The smaller the subset of users used for training and testing, the larger the Probe10 RMSE and the number of epochs. This means that time-complexity of MF is sublinear in the number of users, which is proportional to the number of ratings, the ratio is 209 for the Netflix dataset.

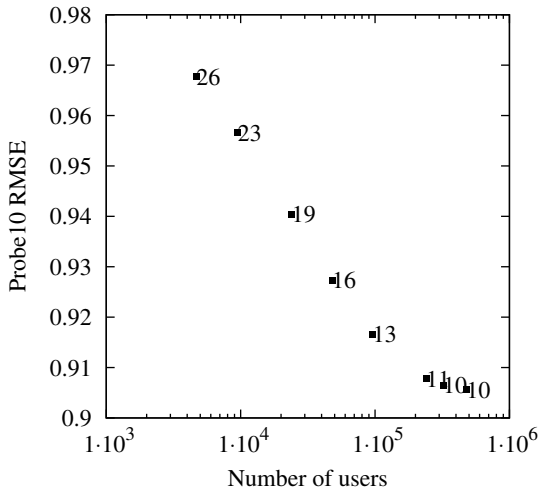

Figure 1: Effect of the number of users on Probe10 RMSE and on the optimal number of training epochs.

4.2.4 Retraining user features

We investigated with three parameter settings the effect of retraining user features: BRISMF#1, and the following two MFs:

- BRISMF#250: $K = 250$, $w_{\underline{u}} = -0.01$, $w_{\overline{u}} = -0.006$, $w_{\underline{m}} = -0.010$, $w_{\overline{m}} = 0.020$, $\eta_u = 0.008$, $\eta_{ub} = 0.016$, $\eta_m = 0.015$, $\eta_{mb} = 0.007$, $\lambda_u = 0.048$, $\lambda_m = 0.008$, $\lambda_{ub} = 0.019$, $\lambda_{mb} = 0, G = 0$.
- BRISMF#1000: the same as BRISMF#250, but $K = 1000$.

The results are summarized on Table 2. Both the simpler case (after the first learning step, reset **U** and retrain only **U**), and the advanced case (after the first learning step, reset **U** and retrain both **U** and **M**) are reported. We append letter "U" to the method name in the simpler case (i.e. BRISMF#1 becomes BRISMF#1U, etc.), and letters "UM" in the advanced case (BRISMF#1UM, etc.). We report the number of epochs required both in the first and the second training procedure (if available). Note, that in case of BRISMF#250 and BRISMF#1000, the retraining of user features greatly improved the performance of those MFs.

Table 2: Examining the effect of retraining user features. We report on Probe10 RMSE and some Quiz RMSE values.

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

4.2.5 Accurate MFs

Here we report on some of the most accurate models that were obtained by the automatic (see above) and manual parameter optimization methods.

- BRISMF#800: manually parameterized MF, with 800 features. Parameter are set to: $K = 800$, $w_{\underline{u}} = -w_{\overline{u}} = w_{\underline{m}} = -w_{\overline{m}} = -0.005, \eta_u = \eta_{ub} = 0.016, \eta_m = \eta_{mb} = 0.005, \lambda_u = \lambda_m = 0.010, \lambda_{ub} = \lambda_{mb} = 0, G = 3.6043$. After 9 epochs, learning rates are multiplied by 0.01, and the model is trained for another 2 epochs.
- SemPosMF#800: this is a semipositive variant of BRISMF#800, where user features are non-negative, item-features are arbitrary. Parameter are set to: $K = 800, w_{\underline{u}} = 0, w_{\overline{u}} = -w_{\underline{m}} = w_{\overline{m}} = 0.005, \eta_u = \eta_{ub} = 0.016, \eta_m = \eta_{mb} = 0.005, \lambda_u = \lambda_m = 0.010, \lambda_{ub} = \lambda_{mb} = 0, G = 3.6043$. After 12 epochs, learning rates are multiplied by 0.01, and the model is trained for another 2 epochs.
- MlMF#200: a BRISMF with 200 features. Automatic parameter optimization.
- MlMF#80: a BRISMF with 80 features. Automatic parameter optimization.
- MomentumMF: a BRISMF with momentum method and 50 features, manually optimized: $K = 50$, $\eta = 0.01$, $\sigma = 0.3$ and $\lambda = 0.00005$. Learnt in 5 epoch.

Table 3: Probe10 RMSE of accurate MFs without and with applying Q-correction and neighbor corrections (S1 and S2). At column S1 and S2 we also indicated the optimal value of parameter α

#	Model	basic	Q	S1	S2	Q+S1+S2	Combination	basic	Q+S1+S2
1	BRISMF#800	0.8940	0.8930	$0.8916_{(\alpha=8)}$	$0.8914_{(\alpha=7)}$	0.8902			
2	SemPosMF#800	0.8950	0.8941	$0.8916_{(\alpha=8)}$	$0.8913_{(\alpha=5)}$	0.8900	1+2	0.8923	0.8880
3	MlMF#200	0.9112	0.9106	$0.9087_{(\alpha=8)}$	$0.9085_{(\alpha=6)}$	0.9076	1+2+3	0.8913	0.8872
4	MlMF#80	0.9251	0.9240	$0.9104_{(\alpha=9)}$	$0.9072_{(\alpha=2)}$	0.9058	1+2+3+4	0.8909	0.8863
5	BRISMF#1000UM	0.8921	0.8918	$0.8905_{(\alpha=7)}$	$0.8907_{(\alpha=5)}$	0.8901	1+2+3+4+5	0.8895	0.8851
6	MomentumMF	0.9031	0.9020	$0.8979_{(\alpha=6)}$	$0.8956_{(\alpha=3)}$	0.8949	1+2+3+4+5+6	0.8889	0.8838

We refer to a variant of the transductive MF algorithm as Q-correction: in eq. (19) to improve predictions we use only the ratings in the Qualify set, not in the Probe10+Qualify set. See Table 3 for the RMSE values of the each method and their blended versions. We applied two neighborhood corrections on the MF models, with similarities S1 and S2.

The Q, S1, S2 and Q+S1+S2 columns represents results of linear regression of multiple columns (2, 2, 2 and 4 respectively) on Probe10 data. The "baisc" column of combinations (1+2, 1+2+3, ..., 1+2+3+4+5+6) are combinations of 2, 3, ..., 6 columns, resp. The Q+S1+S2 column of combinations (1+2, 1+2+3, ..., 1+2+3+4+5+6) are combinations of 8, 12, 16, 20 and 24 columns, resp. In our experiments, blending means ridge regression against the expected Probe10 values.

One can observe in Table 3 that NB correction improves significantly the result of MF based methods. Starting from an average MF (MlMF#80) the reduction of RMSE can be 0.0179, it can reduce the RMSE of the good Momentum MF by 0.0075, and it even improves slightly (0.0026) the very accurate BRISMF#800. In comparison, BellKor's approach ([2], Table 2) results in 0.0096 RMSE reduction, starting from MF with 0.9167 RMSE, here the reduced RMSE score is almost identical with our NB corrected MlMF#80 that has originally only RMSE 0.9251.

If we put in all MFs and all corrections, then the combination yields an RMSE = 0.8838. However, it brings only insignificant improvements if one applies Q-correction technique for all MFs. We get RMSE = 0.8839 if we exclude the Q-corrections of all MFs but the first from the combination. Moreover, if we apply neighbor and Q-correction only on the first MF, then the RMSE increases only to 0.8850. In general, we can state that one "correction technique" brings major decrease in the RMSE when applied only to a single method in the linear combination. If we apply it multiple times, the improvement becomes less. In other words, Q-corrections or neighbor corrections captures the same aspects of the data, regardless of the MF behind them.

4.2.6 *Speed vs. accuracy*

It is interesting and important to investigate the relation of accuracy and speed. We run many randomly parameterized MFs with $K = 40$, and collected the best accuracies in each epoch. Table 4 summarizes the results. A Probe10 RMSE of 0.9071 can be achieved within 200 seconds (including the time to train with the 100 million available ratings and evaluate on the Probe10)! Netflix's Cinematch algorithm can achieve Quiz RMSE = 0.9514.

To our best knowledge, the running times presented here are far more advantageous than of any other methods published in the literature of CF—though authors usually do not tend to publish running time data. This property pairs with very accurate models which makes the presented models and their implementation to be the most favorable matrix factorization approaches.

Table 4: Probe10 RMSE of fast and accurate MFs.

Epoch	Training Time (sec)	RMSE
1	120	0.9188
2	200	0.9071
3	280	0.9057
4	360	0.9028
5	440	0.9008
6	520	0.9002

4.3 RMSE values reported by other authors

We compare the presented Probe10 RMSE values (which differ from Quiz RMSE values at most by 0.0003) with other RMSE values reported for the Netflix Prize dataset.

Authors often report on Probe RMSE or Quiz RMSE values. Probe RMSE values are calculated by leaving out the Probe set from the Train set, while Quiz RMSE is often computed by incorporating the Probe data into the training of the predictor. Thus, Probe RMSE is often much lower than Quiz RMSE.

Paterek in [9] introduces the use of bias features. He reports on Quiz RMSE = 0.9070 for his biased regularized MF (called there RSVD2). Salakhutdinov and Mnih present a Probabilistic MF approach and its variants in [11]. The best result they publish is Quiz RMSE = 0.8970, which is obtained as a combination of 3 MFs. Bell and Koren in [2] provides a detailed description of their alternating least squares approach proposed to matrix factorization. Their matrix factorization approach uses a specialized quadratic optimizer instead of ridge regression to assure non-negative weights. They report on Probe RMSE = 0.9167 for their MF, and their methods need to run either ridge regression or a specialized quadratic optimizer for $(|I|+|J|)/2 \cdot n^*$ times, on $|\mathcal{R}| \cdot n^*$ data with K input variables in total, where n^* is the number of epochs, which is "few tens". Thus, their method requires $\Omega((|I| + |J|) \cdot K^3/2 + |\mathcal{R}| \cdot K^2) \cdot n^*$ steps using either ridge regression or the specialized quadratic optimizer. They report on Quiz RMSE = 0.8982 for their neighbor method executed on the residuals of their MF.

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

Here we neglected the cost of parameter optimization.

Our MF has 13 parameters. We perform the parameter optimization process (Sec. 4.1) with a subset of users (1/6), and with a small K value (typically K is 20 or 40). The optimization process requires 100–200 MF runs. In case of SemPosMF#800, which is manually parameterized, we performed $\sim$50 runs. One may argue that parameter optimization for alternating least squares type MF is faster, since there are no learning rates, thus it has just 9 parameters. We observed that the more parameters the MF have, the easier to tune the parameters to get the same Probe10 RMSE. Consequently, we introduced many parameters, for example $\eta_u, \eta_m, \eta_{ub}, \eta_{mb}$ instead of a single η.

The best results in the NP literature are reported in [3], where the authors briefly describe their approach for the Netflix Progress Prize 2007. They use the MF algorithm presented in [2]. They report only on Quiz RMSE values. Here we summarize the best results of that paper:

- Best stand-alone positive MF: Quiz RMSE = 0.8998
- Best stand-alone positive MF with "adaptive user factors": Quiz RMSE = 0.8955
- Best neighbor method on positive MF: Quiz RMSE = 0.8953

Table 5 compares our best methods with the results reported by [3].

Table 5: Comparison of our best Probe10 and Quiz RMSE values against the Quiz RMSE values reported by [3]

Method	Name of our method	Our Probe10	Our Quiz	Bell et al's Quiz
Simple MF	BRISMF #1000	0.8938	0.8939	0.8998
Tweaked MF	BRISMF #1000UM	0.8921	0.8918	N/A
MF with neighbor correction	BRISMF #1000UM, S1	0.8905	0.8904	0.8953

Clearly, our matrix factorization methods and our hybrid MF-neighbor approaches outperform Bell et al's methods: they are more accurate, much faster (in the $O()$ sense as well), simpler, and can handle the change of users' taste in time.

5. CONCLUSIONS

This paper presented matrix factorization based approaches for rating based collaborative filtering problems. We proposed a few novel MF variants including a fast (semi-)positive MF, a momentum based MF, a transductive MF, and an incremental variant of MF. The neighbor based correction of MF with two similarity functions was also presented. We reported on the RMSE scores of our algorithms evaluated on the Netflix Prize dataset. We have shown that the presented models outperform the already published ones significantly. It has been also pointed out that our algorithms are very effective in terms of time requirement, needing in general only a few minutes to train reasonably accurate models.

6. REFERENCES

[1] R. Bell, Y. Koren, and C. Volinsky. Chasing $1,000,000: How we won the netflix progress prize. *ASA Statistical and Computing Graphics Newsletter*, 18(2):4–12, December 2007.

[2] R. M. Bell and Y. Koren. Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights. In *Proc. of ICDM, IEEE International Conference on Data Mining*, 2007.

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

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

[5] J. Bennett and S. Lanning. The Netflix Prize. In *Proc. of KDD Cup Workshop at SIGKDD'07, 13th ACM Int. Conf. on Knowledge Discovery and Data Mining*, pages 3–6, San Jose, CA, USA, 2007.

[6] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proc. of UAI'98, 14th Conference on Uncertainty in Artificial Intelligence*, pages 43–52. Morgan-Kaufmann, 1998.

[7] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

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

[9] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proc. of KDD Cup Workshop at SIGKDD'07, 13th ACM Int. Conf. on Knowledge Discovery and Data Mining*, pages 39–42, San Jose, CA, USA, 2007.

[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 and A. Mnih. Probabilistic matrix factorization. In J. C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, Cambridge, MA, 2008. MIT Press.

[12] 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.

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

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

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 $\mathbf{R} = [r_{ui}]_{1 \leq u \leq m; 1 \leq i \leq n}$ 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

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

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

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

[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

$$\mathbf{C} = \mathbf{P}^T \mathbf{Q}. \tag{4}$$

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

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

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

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

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

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

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

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

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

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]:

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

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.