Incremental Matrix Factorization for Collaborative Filtering

Patrick Ott
Department of Computer Science
Anhalt University of Applied Sciences
D-06366 Köthen (Saxony-Anhalt), Germany
ott@comp.leeds.ac.uk

Abstract

Based on Singular Value Decomposition an incremental and iterative Matrix Factorization method for very sparse matrices is presented. Such matrices arise in Collaborative Filtering (CF) systems, like the Netflix system. This paper shows how such an incremental Matrix Factorization can be used to predict ratings in a CF system and therefore how to fill the empty fields of a rating matrix of a CF system. Also the here presented method is easy to implement and offers, if implemented in the right way, a good and reliable performance.

I. Introduction

A. Recommendation Systems

Over the past decade and especially since the rapid development of the so called "Web 2.0", recommendation (or recommender) systems based on user opinions gained a high popularity. Basically there are two types of such systems. The content based approach creates and updates profiles of each user and object in the database [16]. Out of those profiles an algorithm is created that helps to give recommendations to users. User profiles could include a lot of information but mostly only those the general user is willing to give to the system, such as demographic information.

The alternative strategy is called Collaborative Filtering (CF), which relies only on past behavior of users and does not require any profiles for users or objects in the system. Such a system has the obvious advantage that it does not need external data, such as the mentioned demographic information. This means that there is firstly no need to collect such data (which is time consuming) and secondly there is also no need to bother the user with it. Also, the CF approach allows to uncover patterns in the user behavior that a content based approach might not be able to discover.

The idea of users giving recommendations to other users or, like most of the times, to a group of users is weakly connected to the topic of Human Computation [21] [23] [22]. In Human Computation the uniqueness of the human being and way of thinking is used to break tasks today's computers and algorithms can not solve efficiently on their own. In a CF system humans are used to form global opinions about objects in the system. Therefore it is possible to treat CF systems as a replacement for content based systems. However, combinations of both systems can prove usefull [3] [13] [2] [17] [15].

In this paper the author concentrates on the so-called explicit CF systems. Such a system is dependend on the users rating the different objects of the recommendation system. On the other hand, an implicit system does not depend on ratings but only on the information which items a user ordered, purchased or looked at.

B. The Netflix Prize

Netflix[1] is the largest online DVD rental service in the United States of America. They claim their portfolio consists of over 80000 titles on 42 million DVDs. The service has 6.8 million subscribers. Those

[1] `http://www.netflix.com`

users gave more than 1.7 billion ratings for the movies in the portfolio. On average a member rates 200 movies [2].

The Netflix Prize[3] [5] is a competition that was launched by Netflix in the year 2005. The task is to learn from a training dataset of approximately 100 million ratings that nearly half a million users (480189 to be exact) gave to 17770 movies and finally predict another 2.8 million ratings. Those ratings are given as a quadruplet consisting of the user (an integer ID), the movie (an integer ID), the date of the rating (day-precise) and the rating itself (an integer from 1 to 5). As an error measurement the Root Mean Square Error (RMSE) is used. Netflixs own forecasting algorithm, which is called Cinematch, is able to give a RMSE of 0.9525. Any algorithm that is capable of improving this result by 10% (which is a RMSE $\leq$ 0.8563) will be considered for the $\$1000000$ grand prize.

II. Mathematical Model

The set $U = \{u_1, u_2, ..., u_p\}$ defines the users of the CF system and the set $M = \{m_{1,}m_2, ..., m_q\}$ is holding the available movies. For the Netflix Prize dataset $p = 480189$ users and $q = 17770$ movies are available. The given set T is called the training set:

$$T \subset U \times M \times O$$

with $O = \{1, 2, 3, 4, 5\}$. The set T defines all known ratings by a user and a movie and the rating the user gave to the movie. Also the dataset given by the Netflix Prize provides the participants with a date, which we do not make any use of.

The function e explains the mapping of a user $u_i \in U$ and a movie $m_j \in M$ to a rating r_{ij}.

$$e : U \times M \to O$$

$$e\left(u_i, m_j\right) = r_{ij}$$

The mapping e is estimated by a model $\hat{e}$ that predicts ratings for a given user / movie pair. The model $\hat{e}$ itself, also called an estimator, is linked with the true mapping e by

$$\hat{e}\left(u_i, m_j\right) = e\left(u_i, m_j\right) + Z$$

where Z is the error between the prediction and the actual rating. Often it is supposed that Z is normal distributed,

$$Z \sim N\left(\mu, \sigma^2\right)$$

with mean μ and variance σ^2.

The rating matrix R, given by

$$R = \left(r_{ij}\right)_{i=1,\dots,p,j=1,\dots,q}$$

is the complete rating matrix, meaning that there are no empty cells in it. This matrix usually is unknown because it is unlikely that in a recommendation system each user rated every single movie. Usually we might only know a subset of this matrix, like we do in the Netflix Prize.

As we already stated the goal of a recommendation system is to recommend new items a user might be interested in. To do so an estimator is required, that is able to learn from given ratings to predict ratings that have not been made yet. In this way a possible high rating of a user for a movie can be predicted and for this reason the system can recommend this movie to the user.

Netflix provides the participants of the Prize with a qualification set $Q \subset U \times M$. This set holds 2.8 million unknown ratings and on its basis any prediction algorithm developed by a participant is compared

[2]http://www.netflix.com/MediaCenter?id=5379&hnjr=8#facts

[3]http://www.netflixprize.com/

to Netflix prediction system CineMatch. Since the Netflix Prize dictates the Root Mean Square Error (RMSE) as the tool for error measurement we can obviously define the goal of the prize to minimize this RMSE on the qualification set Q. Furthermore it will satisfy to only minimize the quadratic part of the RMSE, since a minimization of this part will also result in the minimization of the complete RMSE formula.

$$\sum_{(u_i,m_j)\in Q} \left(\hat{e}\left(u_i,m_j\right)-e\left(u_i,m_j\right)\right)^2 \to \min$$

To train $\hat{e}$ it requires a good setup of $\hat{e}$. This setup could be found by minimizing the RMSE of the training set with respect to some parameters $\hat{e}$ makes use of. However, using the training set for validation as well will lead to overtraining. Therefore we remove a validation set V from T, $V \subset T$ and we use $T \setminus V$ for training. Now we can define the learning algorithm clearly by

$$\sum_{(u_i,m_j,r_{ij})\in V} \left(\hat{e}\left(u_i,m_j\right)-r_{ij}\right)^2 \to \min \tag{1}$$

while $\hat{e}$ is training on the examples of $T \setminus V$. We can treat equation 1 as a second objective function next to the optimization problem of

$$\sum_{(u_i,m_j,r_{ij})\in T\setminus V} \left(\hat{e}\left(u_i,m_j\right)-r_{ij}\right)^2 \to \min \tag{2}$$

In such a setup equation 1 has a higher priority than the objective function of equation 2. Meaning that in case the maximum performance on V (the smallest possible RMSE) is reached training stops although it might still be possible to minimize the performance on $T \setminus V$.

III. Incremental Matrix Factorization

The usage of Matrix Factorization (MF) for the Netflix Prize was firstly suggested by Brandyn Webb (a.k.a. Simon Funk) in his journal[1]. MF is strongly connected to Singular Value Decomposition (SVD) [1] [11] [4] [8] [9] [10].

The basic idea behind the classical SVD-approach is to decompose a given rating matrix P of size $n \times m$ into a product of three matrices,

$$P = A\Sigma B^T$$

where A is of size $n \times n$, Σ of size $n \times m$ and B of size $m \times m$. Both matrices, A and B, are orthogonal matrices. The matrix Σ is a diagonal matrix with k non-zero entries. Therefore the effective dimensions of the three matrices above are $n \times k$, $k \times k$ and $k \times m$. These k diagonal entries ς_i of the matrix Σ are all positive with $\varsigma_1 \geq \varsigma_2 \geq ... \geq \varsigma_k > 0$. The columns of A are called the left singular vectors of P. They are orthonormal eigenvectors of AA^T. The columns of the matrix B are called the right singular vectors of P (the orthonormal eigenvectors of A^TA).

If we now retain only the $r \ll k$ greatest singular elements, the product of those three matrices will be a matrix of rank r, namely $\hat{P}$. Furthermore this matrix is the closest rank-r approximation of P in terms of the Frobenius Norm.

$$\left\|\hat{P}-P\right\|_F^2 = \sum_{i=1}^{n}\sum_{j=1}^{m} \left|\hat{p}_{ij}-p_{ij}\right|^2$$

SVD is just one of the many decomposition techniques for matrices of rectangular size [19] [7].

Netflix is providing the participators of the Prize with a matrix R that is very sparse, it has many unknown values. So the classical SVD-approach will only work with some tricks, like replacing the

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

unknown elements with zeros or using other more sophisticated methods [12] [6] [18]. By using incremental learning (which should be understood as learning data instance by data instance) this problem can be avoided.

A. The Base Algorithm

The technique presented here is very similar to the basics of SVD. Instead of decomposing R into three matrices we only focus on two matrices, called the feature-matrices.

$$R \approx SV = \hat{R}$$

where S now is a $p \times f$ matrix, and V is a $f \times q$ matrix. f is the number of used features. Those features desribe a user (matrix S) or a movie (matrix V). User u_i is described by row $\underline{s}_i^T$ of matrix S and each movie m_j is described by a column $\underline{v}_j$ of matrix V. For example we can think of a feature "action" that identifies how much action a movie flavors and how much a given user likes action-movies. A single prediction of a rating of a movie m_j by user u_i is then given by the dot product of the feature vectors of the user u_i and the movie m_j

$$\hat{e}_{MF}\left(u_i, m_j\right) = \underline{s}_i^T \underline{v}_j = \sum_{k=1}^{f} s_{ik} v_{kj}$$

where s_{ik} and v_{kj} are the components of the vectors $\underline{s}_i$, $\underline{v}_j$. Now we can write out the quadratic error between the estimated rating and the real rating by

$$\varepsilon_{ij}^2 = \left(r_{ij} - \hat{r}_{ij}\right)^2 = \left(r_{ij} - \sum_{k=1}^{f} s_{ik} v_{kj}\right)^2$$

To minimize this error, $\varepsilon_{ij}^2 \rightarrow \min$, we differentiate the equation with respect to the different features and obtain the following update equations

$$s'_{ik} = s_{ik} + \alpha \frac{\partial \varepsilon_{ij}^2}{\partial s_{ik}} = s_{ik} + 2\alpha \varepsilon_{ij} v_{kj} \qquad v'_{kj} = v_{kj} + \alpha \frac{\partial \varepsilon_{ij}^2}{\partial v_{kj}} = v_{kj} + 2\alpha \varepsilon_{ij} s_{ik}$$

whee α is the learning rate. This number controls how much of the errors slope is added to the new feature value. The value α should be rather small, around 0.001. With the help of the update equations the basic learning algorithm is defined as follows

Algorithm 1 Base Algorithm

```
1: Initialize the features matrices $S$ and $V$ for the first time, either by a fixed number or randomly.
2: for $k$ = 1 to $f_{max}$ do
3:   repeat
4:     for $\forall \left(u_i, m_j, r_{ij}\right) \in T$ do
5:       Compute $\varepsilon_{ij}$.
6:       Update $s_{ik}, v_{kj}$.
7:     end for
8:     Recompute $\sum_{(u_i,m_j,r_{ij}) \in T \setminus V} \varepsilon_{ij}^2$ and $\sum_{(u_i,m_j,r_{ij}) \in V} \varepsilon_{ij}^2$
9:   until some terminal condition is met
10: end for
```

B. Variations of the Base Algorithm

There are plenty of variations of this algorithm. One might consider training one feature at a time while another one might consider training all features simultaneously. It turns out that the later method converges faster in terms of required iterations. However, in our implementation we noticed that, though the need of more iterations over all, the first method works faster because there are several ways to tune the programming code by caching already calculated features. One advantage of the simultaneous training is that the variance of each feature does not decline as it does when each feature is trained independently. This means that the first feature already covers a huge part of the prediction while the later features only try to predict smaller residuals.

C. Linear modifications to the method of iterative MF

1) Regularization parameters: A few additions to the iterative MF have been proposed. One of them includes adding regularizations with a factor β to the update equations. This addition will partially suppress overtraining and therefore improve performance on unseen examples. The update equations change to

$$s'_{ik} = s_{ik} + \alpha \left(\frac{\partial \varepsilon_{ij}^2}{\partial s_{ik}} - \beta s_{ik} \right) = s_{ik} + \alpha \left(2\varepsilon_{ij} v_{kj} - \beta s_{ik} \right)$$

$$v'_{kj} = v_{kj} + \alpha \left(\frac{\partial \varepsilon_{ij}^2}{\partial v_{kj}} - \beta v_{kj} \right) = v_{kj} + \alpha \left(2\varepsilon_{ij} s_{ik} - \beta v_{kj} \right)$$

The equivalent[2] error equation for these update equations is

$$\varepsilon_{ij}^2 = \left(r_{ij} - \sum_{k=1}^{n} s_{ik} v_{kj} \right)^2 + \frac{\beta}{2} \sum_{k=1}^{n} \left(s_{ik}^2 + v_{kj}^2 \right) = \left(r_{ij} - \sum_{k=1}^{n} s_{ik} v_{kj} \right)^2 + \frac{\beta}{2} \left(\|\underline{s}\|^2 + \|\underline{v}\|^2 \right)$$

We call $\beta \frac{1}{2} \sum_{k=1}^{f} \left(s_{ik}^2 + v_{kj}^2 \right)$ the regularization term. Since the variance of the features is declining during training, the regularization parameter βs_{ik} respectively βv_{kj} are also declining with each new feature trained. Meaning that those additions introduce an artificial error to the update equation, especially for the first features with a very high variance. This error declines over time. To put it simple it moves some weight from the earlier features to the later ones and therefore the algorithm can train longer and suppress overtraining a bit.

2) Linear Matrix Factorization: Paterek [14] suggested to add biases to the prediction model. He proposed to include two constants, one for each movie and one for each user, in the equation of $\hat{e}$. Those constants are trained simultaneously. We propose a slightly different modification of $\hat{e}$. For each feature we implement two constants, like proposed by Paterek, and train them together with the feature matrices. This method is called Linear Matrix Factorization (LMF)

$$\hat{e}_{LMF} \left(u_i, m_j \right) = \sum_{k=1}^{f} s_{ik} v_{kj} + c_i + d_j$$

where the weights c_i and d_j are trained simultaneously with the movie- and userfeatures. The update equations for the elements of the feature matrices stay unchanged. The update equations for c_i and d_j are given by

$$c'_i = c_i + \chi \left(\frac{\partial \varepsilon_{ij}^2}{\partial c_i} - \delta c_i \right) = c_i + \chi \left(2\varepsilon_{ij} - \delta c_i \right)$$

[2]The error equation is equivalent in terms of that its derivation will result in a gradient, of which each component will be equivalent to the here presented update equations.

$$d_j' = d_j + \chi \left(\frac{\partial \varepsilon_{ij}^2}{\partial d_j} - \delta d_j \right) = d_j + \chi \left(2\varepsilon_{ij} - \delta d_j \right)$$

χ is the learning rate for the constants, while δ is another regularization parameter. The final error for a given rating and its prediction by the estimator including all regularization parameters is

$$\varepsilon_{ij}^2 = \left(r_{ij} - \sum_{k=1}^{n} s_{ik} v_{kj} + c_i + d_j \right)^2 + \frac{\beta}{2} \left(\|\underline{s}\|^2 + \|\underline{v}\|^2 \right) + \frac{\delta}{2} \left(c_i^2 + d_j^2 \right)$$

D. The final algorithm

The dataset of the Netflix Prize has some idiosyncrasies that need to be considered in form of changing the algorithm to perform well. One improvement is to define a minimum number of training epochs r_{min} per feature as well as a maximum number of training epochs r_{max}. Those two limits are dynamically adjusted depending on the number of the feature that is trained. This alternation of the main algorithm is necessary because the algorithm would otherwise stop training just after the first few epochs. Especially for the later features it requires a lot of epochs to actually come to the point where the RMSE on the validation set improves again. It is nearly as if the algorithm would have to walk a certain way before it can approach the next local (or global) minimum. For the minimum number of training epochs $r_{min} = 75 + f$ is used while the maximum epochs are set to $r_{min} = 85 + 2f$. f is the actual trained feature.

Validation is done on a smaller validation set with roughly 100000 instances. This is enough to ensure that the RMSE is precise enough and furthermore training is not slowed down significantly. In case the minimum number of training epochs for this feature is crossed and the RMSE on the validation set is still declining, training on this feature stops and the algorithm moves on to the next feature. The improvement of RMSE is measured within five steps, meaning that for the RMSE on the validation set of the i'th feature and the j'th epoch, $RMSE_i^j$, the given RMSE difference is

$$\Delta_i^j = RMSE_i^{j-5} - RMSE_i^j$$

A minimum of five training epochs for each feature is required. The final training algorithm is presented in Algorithm 2.

Algorithm 2 Base Algorithm

```
1:  Initialize the features matrices $S$ and $V$ for the first time, $S$ and $V$ to $0.1$ and $\underline{c}$ and $\underline{d}$ to $0.9$.
2:  for $k$ = 1 to $f_{max}$ do
3:      $r_{max} = 75 + f, r_{min} = 85 + 2f$
4:      for $r$ = 1 to $r_{max}$ do
5:          for $\forall (u_i, m_j, r_{ij}) \in T$ do
6:              Compute $\varepsilon_{ij}$.
7:              Update $s_{ik}, v_{kj}, c_i, d_j$.
8:          end for
9:          Recompute $\left( \sum_{(u_i,m_j,r_{ij}) \in T \setminus V} \varepsilon_{ij}^2 \right)^{\frac{1}{2}}$ and $RMSE_k^r = \left( \sum_{(u_i,m_j,r_{ij}) \in V} \varepsilon_{ij}^2 \right)^{\frac{1}{2}}$
10:         Recompute $\Delta_i^j$.
11:         if $\Delta_i^j < 0 \wedge r > r_{\min}$ then
12:             break
13:         end if
14:     end for
15: end for
```

TABLE I

RMSEs FOR INCR MF. $r_{max} = 85 + 2f$, $r_{min} = 75 + f$, $\alpha = 2.5 \times 10^{-4}$, $\chi = 10^{-4}$

MF preferences	$f = 32$	$f = 64$	$f = 96$	$f = 128$	$f = 164$
$\beta, \delta = 0.015$	0.8168	0.8105	0.8081	0.8070	0.8072
$\beta, \delta = 0.02$	0.8170	0.8095	0.8068	0.8053	0.8043
$\beta, \delta = 0.0225$	0.8181	0.8104	0.8075	0.8059	0.8049
$\beta, \delta = 0.025$	0.8186	0.8113	0.80910	0.8069	0.8054
$\beta, \delta = 0.03$	0.8215	0.8152	0.8137	0.8116	0.8091

IV. Experiments

A precise performance measurement of different variations of incremental MF is ensured by removing a set of roughly 3 million ratings from the given training set of the Netflix Prize. Of this set of 3 million rating instances 90463 ratings were shaped off. Those were used for validation of the MF techniques and to determine when to stop training a feature (set V_{MF}). The total size of the validation set V therefore reduces to 2924999. The precise size of the training set T is 97465045.

Table I gives the RMSE of some examples of implementations of the MF algorithm. The setup used for all instances of incremental learning consists of a fixed learning rate of $\alpha = 2.5 \times 10^{-4}$ for the feature matrices S and V and $\chi = 10^{-4}$ for the constant vectors $\underline{c}$ and $\underline{d}$. The regularization parameters for the matrices and vectors β, δ are given in the range of 0.015 and 0.03. A maximum of 164 features were trained. The maximum number of iterations was set to $85 + 2f$ while the minimum number was set to $75 + f$ and f is the number of the actual trained feature.

Figure IV shows the declining error of the implementations on the validation set that is used to determine when training should stop. As can be seen regularization parameters of $\beta, \delta = 0.0225$ perform the best. However, the graph also shows that with a higher number of features other regularization parameters, such as $\beta, \delta = 0.025$, could pull ahead. Furthermore it is obvious that the number of minimum training epochs per feature was set too low for the implementation using regularizations parameters of $\beta, \delta = 0.025$ and $\beta, \delta = 0.03$ since there is a horizontal line indicating that training did not improve at all during that time. This is due to the fact that the algorithm did not reach the point, in terms of training epochs per feature, where the RMSE on the validation set actually begins improving. With a higher number of minimum training iterations per feature the implementation using $\beta, \delta = 0.025$ might perform best.

Figure IV shows an interesting picture. While for the validation set (Figure IV) the regularization parameters actually show some effect, they do not do at all on the training set. The higher the parameters β, δ are set the worse the RMSE on the training set is. The observation that the RMSE on the validation set improves when using well set parameters β, δ while it does not improve the performance on the training set obviously points out that those parameters can supress overtraining substantially.

Figure IV plots the RMSE on validation against the RMSE on the training set. The last few epochs were captured because only those final training steps actually differ substantially. It is clearly seen that the regularization parameters have an effect on overtraining and cause an impact on the error of the validation set.

V. Future Work

A. Histogram Shifting and Average Error Normalization

While the predictions of our estimators are given within the interval $[1, 5]$ as real numbers the actual ratings of the Netflix Prize were natural numbers of $\{1, 2, 3, 4, 5\}$ stars. Therefore it might be a valueable idea to shift the histograms of the predictions to make them fit the distributions of the ratings of the training set. However, the question stays how to round the predictions. The publication on the Gravity Recommendation System [20] presented some rounding approaches. None of them turned out to be useful

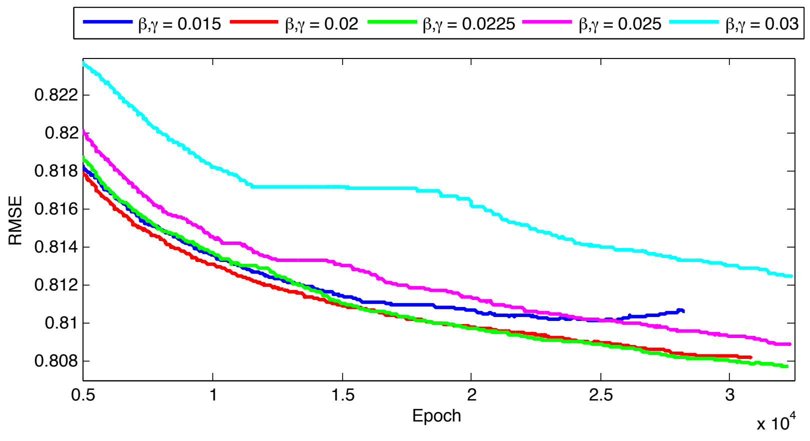

Fig. 1. Incremental MF performance on validation. Epochs from 5000 to 30000. Maximum of 164 features. Learning rates fixed at $\alpha = 0.00025$ and $\gamma = 0.0001$.

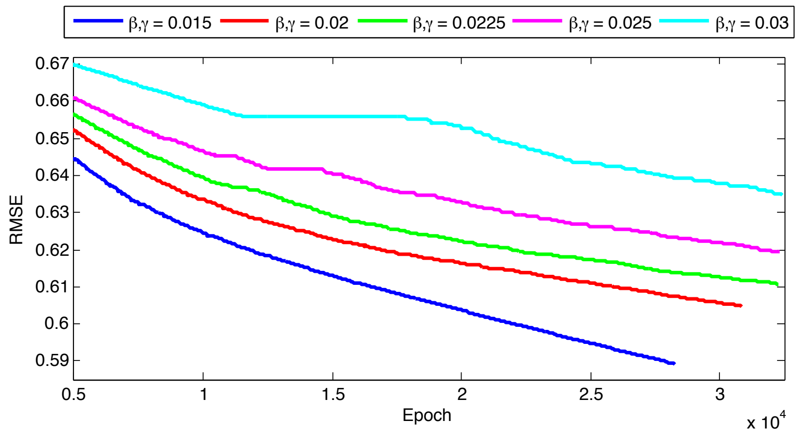

Fig. 2. Incremental MF performance on training set. Epochs from 5000 to 30000. Maximum of 164 features. Learning rates fixed at $\alpha = 0.00025$ and $\gamma = 0.0001$. The higher the regularization parameters, the lower the performance.

for improving the RMSE on their validation set. However, they might still be useful to approach histogram shifting.

One more way to improve performance of the algorithms is a technique we call Average Error Normalization. Figure 4 shows the mean error of each prediction of a MF prediction model. Of course such a plot requires some form of binning of the prediction values. For the plot we used a setup of 50 bins. The predictions of the validation set were arranged in those bins and for each bin the average error was computed. Then each bins upper threshold was plotted against the average error. As can be seen from the plot it is obviously true that the smaller predictions are actually a little too small while the higher predictions are a little too high. By normalizing this plot to the x-axis, meaning to subtract the offset from each item in the bin, we are able to improve the resulting RMSE by a small amount. Furtheremore this normalization could be used in every feature training step of a MF. This means that once a feature finished training the predictions of that feature (and the features before it) are not only clipped to the $[1, 5]$ interval but also normalized in terms of the average error per bin.

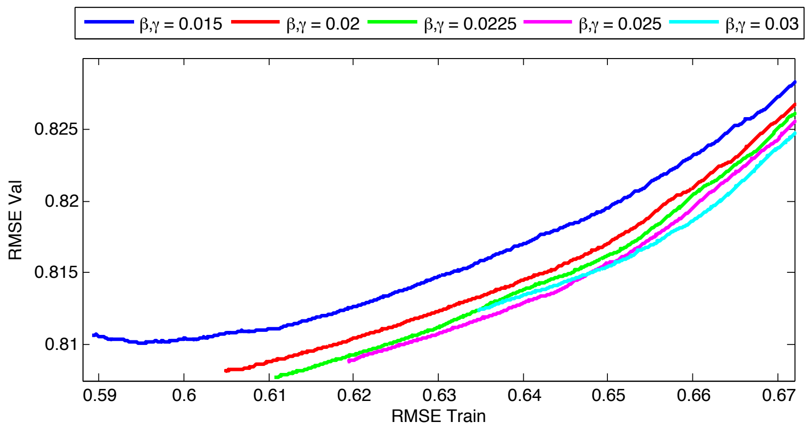

Fig. 3. Approximately the final 5000 training epochs of different incremental MFs. RMSE on training set plotted against the RMSE on the validation set.

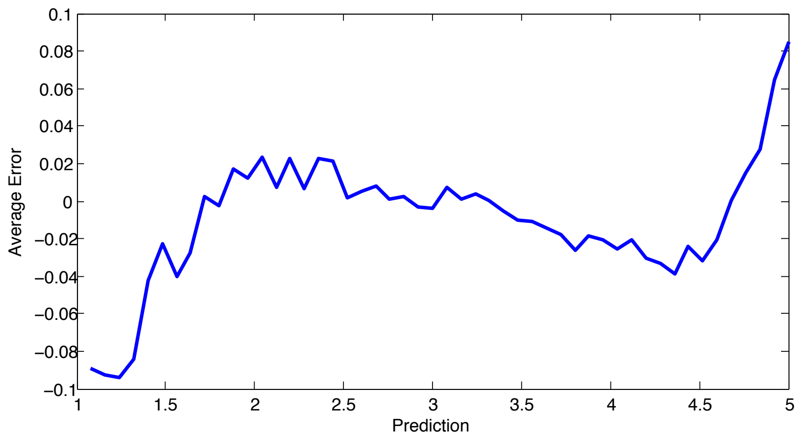

Fig. 4. Actual predictions of validation set plotted against average prediction error

B. Non-linear Estimators

Due to the high complexity of the Netflix Prize dataset it is only likely that a simple linear approach, like presented in this paper, will not be able to fully cover the characteristics of the dataset. Therefore it might be a valueable idea to implement and research non-linear estimators. Since the here presented estimators already make use of the dot product it is a common conclusion to move on to the usage of kernel-based methods. Especially the polynomial kernels are noteworthy because they directly include the dot product, which is already being computed in the linear appoach.

$$\hat{e}\left(u_i, m_j\right) = \left(\underline{s}_i^T \underline{v}_j + 1\right)^z + c_i + d_j$$

with $z \in \mathbb{R}$. Early experiments have shown that a kernel-based approach can keep up with linear approaches up to a certain point. However, the problem emerged that kernel-based methods overtrain very quickly, which caused the training RMSE to drop further and further but the RMSE on the validation set to stagnate.

VI. Conclusion

A method to incrementally decompose the rating matrix R of a CF system was presented. Whilst the method is based on the algorithm of steepest descent it does not directly make use nor compute the gradients usually used by such methods. A short performance evaluation was given, which prooved that regularization parameters have an effect on the final RMSE and also are able to reduce overtraining significantly. The here presented method is easy to implement and due to its simplicity also fast performing.

Acknowledgment

The author would like to thank the Data Miners Inc., especially Michael J.A. Berry, for providing the opportunity to do an internship at the company. This work was mainly done at the Data Miners Inc. in Boston, MA. Also the author would like to thank Dr. Bernd Krause from the Anhalt University for the vital input and helpful corrections.

References

[1] Herve Abdi. Singular value decomposition (svd) and generalized singular value decomposition (gsvd). In Neil Salkind, editor, *Encyclopidia of Measurement and Statistics*. Thousands Oaks, CA, 2007.

[2] Joshua Alspector, Aleksander Kolcz, and Nachimuthu Karunanithi. Comparing feature-based and clique-based user models for movie selection. In *DL '98: Proceedings of the third ACM conference on Digital libraries*, pages 11–18, New York, NY, USA, 1998. ACM.

[3] Marko Balabanovi and Yoav Shoham. Fab: content-based, collaborative recommendation. *Commun. ACM*, 40(3):66–72, 1997.

[4] David Bau III and Lloyd N. Trefethen. *Numerical linear algebra*. Society for Industrial and Applied Mathematics, 1997.

[5] James Bennett and Stan Lanning. The netflix prize. In *Proceedings of the KDD Cup 2007*, pages 3–6. ACM Press, 2007.

[6] Michael W. Berry. Large scale sparse singular value computations. Technical report, Department of Computer Science, University of Tennessee, 107 Ayres Hall Knoxville TN, 37996-1301, 2002.

[7] Antonio J. Conejo, Enrique Castillo, Roberto Minguez, and Raquel Garcia-Bertrand. *Decomposition Techniques in Mathematical Programming: Engineering and Science Applications*. Springer, 2006.

[8] J. Demmel and W. Kahan. Accurate singular values of bidiagonal matrices. *SIAM J. Sci. Statist. Comput.*, 11(5):873–912, 1990.

[9] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[10] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, 1996.

[11] Gene H. Golub and C. Reinsch. Singular value decomposition and least squares solutions. *Numerische Mathematik*, 14(5), 1970.

[12] Miklo Kurucz, Andras A. Benczur, and Kroly Csalogny. Methods for large scale svd with missing values. In *Proceedings of the KDD Cup 2007*, pages 54–58. ACM Press, 2007.

[13] Prem Melville, Raymond J. Mooney, and Ramadass Nagarajan. Content-boosted collaborative filtering for improved recommendations. Technical report, Department of Computer Sciences, University of Texas, Austin, TX 78712, 2002.

[14] Arkadiush Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of the KDD Cup 2007*, pages 39–42. ACM Press, 2007.

[15] Michael J. Pazzani. A framework for collaborative, content-based and demographic filtering. *Artificial Intelligence Review*, 13(5-6):393–408, 2004.

[16] Michael J. Pazzani and Daniel Billsus. Content-based recommendation systems. *Lecture Notes in Computer Science*, 4321:325–341, 2007.

[17] Paul Resnick and Hal R. Varian. Recommender systems. *Commun. ACM*, 40(3):56–58, 1997.

[18] Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. Technical report, Department of Computer Science, University of Toronto, Kings College Rd, M5S 3G4, Canada, 2007.

[19] David Skillicorn. *Understanding Complex Datasets: Data Mining with Matrix Decompositions*. Chapman and Hall/CRC, 2007.

[20] Gabor Takacs, Istvan Pilaszy, Bottyan Nemeth, and Domonkos Tikk. On the gravity recommendation system. In *Proceedings of the KDD Cup 2007*, pages 22–30. ACM Press, 2007.

[21] Luis von Ahn, Manuel Blum, and John Langford. Telling humans and computers apart automatically. *Commun. ACM*, 47(2):56–60, 2004.

[22] Luis von Ahn and Laura Dabbish. Labeling images with a computer game. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 319–326, New York, NY, USA, 2004. ACM Press.

[23] Luis von Ahn, Ruoran Liu, and Manuel Blum. Peekaboom: a game for locating objects in images. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 55–64, New York, NY, USA, 2006. ACM Press.