Improving Maximum Margin Matrix Factorization

Markus Weimer, Alexandros Karatzoglou, Alex Smola

Improving Maximum Margin Matrix Factorization

Markus Weimer[1], Alexandros Karatzoglou[2], and Alex Smola[3]

[1] Technische Universität Darmstadt, Germany, weimer@acm.org
[2] INSA de Rouen, LITIS, France, alexis@ci.tuwien.ac.at
[3] NICTA, Canberra 2601, Australia, alex.smola@gmail.com

Abstract. Collaborative filtering is a popular method for personalizing product recommendations. Maximum Margin Matrix Factorization (MMMF) has been proposed as one successful learning approach to this task and has been recently extended to structured ranking losses. In this paper we discuss a number of extensions to MMMF by introducing offset terms, item dependent regularization and a graph kernel on the recommender graph. We show equivalence between graph kernels and the recent MMMF extensions by Mnih and Salakhutdinov [1]. Experimental evaluation of the introduced extensions show improved performance over the original MMMF formulation.

1 Introduction

Collaborative filtering has gained much attention in the machine learning community due to its applications in electronic commerce sites such as those of Amazon, Apple and Netflix. Such sites typically offer personalized recommendations to their customers. The quality of these suggestions is crucial to the overall success, since good recommendations will increase the propensity of a purchase.

However, suggesting the right items is a highly nontrivial task: (1) There are many items to choose from. (2) Customers are willing to consider only a small number of recommendations (typically in the order of ten). Collaborative filtering addresses this problem by learning the suggestion function for a user from ratings provided by this and other users on items.

The known data can be thought of a sparse $n \times m$ matrix Y of rating/purchase information, where n denotes the number of users and m is the number of items. In this context, Y_{ij} indicates the rating of item j given by user i. Typically, the rating is given on a five star scale and thus $Y \in \{0, \ldots, 5\}^{n \times m}$, where the value 0 indicates that a user did not rate an item. In this sense, 0 is special since it does *not* indicate that a user dislikes an item but rather that data is missing.

Related Work A common approach to collaborative filtering is to fit a factor model to the data. For example by extracting a feature vector for each user and item in the data set such that the inner product of these features minimizes an explicit or implicit loss functional (e.g. [2] following a probabilistic approach).

The underlying idea behind these methods is that both user preferences and item properties can be modeled by a number of factors.

The basic idea of matrix factorization approaches is to fit the original Y matrix with a low rank approximation F. More specifically, the goal is to find such an approximation that minimizes the sum of the squared distances between the known entries in Y and their predictions in F. One possibility of doing so is by using a Singular Value Decomposition of Y and by using only a small number of the vectors obtained by this procedure. In the information retrieval community this numerical operation is commonly referred to as Latent Semantic Indexing.

Note, however, that this method does not do justice to the way Y was formed. An entry $Y_{ij} = 0$ indicates that we did not observe a (user,object) pair. It does, however, not indicate that user i disliked object j. In [3], an alternative approach is suggested which is the basis of the method described in this paper. We aim to find two matrices U and M where $U \in R^{n \times d}$ and $M \in R^{d \times m}$ such that $F = UM$ with the goal to approximate the *observed* entries in Y rather than approximating all entries at the same time.

In general, finding a globally optimal solution of the low rank approximation problem is unrealistic: in particular the aproach proposed by [3] for computing the a weighted factorization, which is relavant in collaborative filtering settings, requires semidefinite programming which is feasible only for hundreds, at most, thousands of terms. Departing from the goal of minimizing the rank, Maximum Margin Matrix Factorization (MMMF) aims at minimizing the Froebenius norms of U and M, resulting in a set of convex problems when taken in isolation and thus tractable by current optimization techniques. It was shown in [4, 5] that optimizing the Froebenius norm is a good proxy for optimizing the rank in its application to model complexity control. Similar ideas based on matrix factorization have been also proposed in [1, 6].

Recently [7] proposed to extend the general MMMF framework in order to minimize structured (ranking) losses instead of the sum of squared errors on the known ratings. Key in the reasoning is that collaborative filtering is often at the heart of recommender systems. For those, only the *ranking* of unrated items in terms of the user preferences matter. To enable effective optimization of the structured ranking loss, a novel optimization technique [8] was used to minimize the loss in terms of the Normalized Discounted Cumulative Gain (NDCG).

Our Contribution Building upon the results outlined above, we introduce a number of extensions to the these MMMF models.

- An efficient means of computing the gradient in multiclass ordinal regression.
- A bias for movies and users to deal with heterogeneity of movies and users.
- An automatic adaptive regularization scheme which can deal with the varying number of movies per user and likewise users per movie.
- A graph kernel that captures similarities between users and items in the recommender graph in the spirit of [1, 9]. We prove that both methods are essentially equivalent and equal to a graph kernel on the recommender graph.

For each of these extensions, we show how they can be integrated into the MMMF framework such that structured estimation as proposed in [7] is still feasible.

The paper is organized as follows: Section 2 describes the general MMMF model, its generalization to structured estimation and the use of state-of-the-art optimization methods to train the model. Section 3 describes our extensions to that model. In section 4, we discuss our experimental evaluation and section 5 concludes the paper with remarks on future work.

2 Maximum Margin Matrix Factorization

2.1 Optimization Problem

MMMF computes a dense approximation F of the sparse matrix Y which forms the training data. The approximation is based on the modeling assumption that any particular rating of item j by user i is a linear combination of item and user features. Thus, the approximation can be written as $F = UM$. Here, U_{i*} represents the feature vector for user i and M_{*j} is the feature vector for item j. The predicted rating of item j by user i is then the inner product between these feature vectors:

$$F_{ij} = \langle U_{i*}, M_{*j} \rangle$$

Finding the appropriate matrices U and M is achieved by minimizing the regularized loss functional where the Froebenius norm ($\|U\|_F^2 = \operatorname{tr} UU^\top$) of the U and M matrices is used for capacity control and thus overfitting prevention. The Froebenius norm has been introduced to the MMMF framework and shown to be a proper norm on F [5]. This leads us to the following optimization problem:

$$\underset{U,M}{\text{minimize}}\, L(F,Y) + \frac{\lambda_m}{2}\|M\|_F^2 + \frac{\lambda_u}{2}\|U\|_F^2 \tag{1}$$

Here λ_m, λ_u are the regularization parameters for the M and U matrix respectively and $F = UM$. Moreover, $L(F,Y)$ is a loss measuring the discrepancy between Y and F.

The optimization problem can be solved exactly by using a semi definite reformulation [4]. However, this dramatically limits the size of the problem to several thousand users / movies. Instead, we exploit the fact that the problem is convex in U and M respectively when the other variables are fixed to perform subspace descent [10].

2.2 The Loss

Squared Loss In the original MMMF formulation, $L(F,Y)$ was chosen to be the sum of the squared errors [4]:

$$L(F,Y) = \frac{1}{2}\sum_{i=0}^{n}\sum_{j=0}^{m} S_{ij}(F_{ij} - Y_{ij})^2 \text{ where } S_{ij} = \begin{cases} 1 & \text{if user } i \text{ rated item } j \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

This loss decomposes for the non zero elements of Y and consequently it is amenable to efficient minimization by repeatedly solving a linear system of equations for each row / column of U and M separately (i.e. in parallel) — the objective function in (1) is convex quadratic in U and M respectively whenever the other term is fixed.

The gradient of $L(F, Y)$ with respect to F can be computed efficiently, since $\partial_{F_{ij}} L(F, Y) = S_{ij} F_{ij} - Y_{ij}$. This means that we have

$$\partial_F L(F, Y) = S.*(F - Y) \tag{3}$$

where $.*$ implies element-wise multiplication of S with $F - Y$. In other words, the gradient of the loss is a sparse matrix.

Non separable Loss This decomposition into losses, depending on Y_{ij} and F_{ij} alone, fails when dealing with structured losses that take an entire row of predictions, i.e. all predictions for a given user into account. Such losses are closer to what is needed in recommender systems, since users typically want to get good recommendations about which movies they are interested in. A fairly accurate description of which movies they hate is probably less desirable. The recent paper [7] describe an optimization procedure which is capable of dealing with such problems. In general, a non separable loss takes on the following form:

$$L(F, Y) := \sum_{i=1}^{n} l(F_{i*}, Y_{i*}) \tag{4}$$

Gradients of $L(F, Y)$ decompose immediately into $\partial_{F_{i*}} l(F_{i*}, Y_{i*})$. This allows for efficient gradient computation.

Ordinal Loss We now discuss an extension of a common ranking loss, namely the ordinal regression score, as suggested in [11]. For simplicity of notation we only study a row-wise loss $l(f, y)$, where we assume that $f := F_{i*}$ and $y := Y_{i*}$ have already been compressed to contain only nonzero entries in Y_{i*} with the corresponding entries of F_{i*} having been selected accordingly.

Assume that y is of length m containing m_j movies of score j, that is $\sum_j m_j = m$. For a given pair of movies (u, v) we consider them to be ranked correctly whenever $y_u > y_v$ implies that also $f_u > f_v$. A loss of 1 is incurred whenever this implication does not hold. That is, we count

$$\sum_{y_u > y_v} C(y_u, y_v) \{f_u \leq f_v\}. \tag{5}$$

Here $C(y_u, y_v)$ denotes the cost of confusing a movie with score y_u with one of score y_v. Since there are $n = \frac{1}{2}\left[m^2 - \sum_j m_j^2\right]$ terms in the sum we need to renormalize the error by n in order to render losses among different users comparable. Moreover, we need to impose a soft-margin loss on the comparator

Algorithm 1.1. $(l, g) = \mathrm{l}(f, y, C)$

input Vectors f and y, score matrix C
output Loss l and gradient g

initialize $l = 0$ (loss) and $g = 0$ (gradient)
for $i = 1$ **to** n **do**
  $b_i = 0$ (lower counter) and $u_i = m_i$ (upper counter)
end for
Let $c = [f - \frac{1}{2}, f + \frac{1}{2}] \in \mathbb{R}^{2m}$
Rescale $C \leftarrow 2C/(m^2 - \|u\|_2^2)$
index = Argsort(c) (find overlaps between pairs)
for $i = 1$ **to** $2m$ **do**
  $j = \mathrm{index}(i) \bmod m$ and $z = y_j$
  if $\mathrm{index}(i) \leq m$ (from the first half) **then**
    for $k = 1$ **to** $z - 1$ (we should be better than those) **do**
      $l \leftarrow l - C(k, z)u_k c_j$ and $g_j \leftarrow g_j - C(k, z)u_k$
    end for
    $b_z \leftarrow b_z + 1$ (there are now $b_z + 1$ elements below us)
  else
    for (they should be better than us) $k = z + 1$ **to** n **do**
      $r \leftarrow r + C(z, k)b_k c_{j+m}$ and $g_j \leftarrow g_j + C(z, k)b_k$
    end for
    $u_z \leftarrow u_z - 1$ (we've just seen one more term from above)
  end if
end for

$\{f_u \leq f_v\}$ to obtain a convex differentiable loss. This yields the loss

$$l(f, y) = 2\Big[m^2 - \sum_j m_j^2\Big]^{-1} \sum_{y_u > y_v} C(y_u, y_v) \max(0, 1 - f_u + f_v). \tag{6}$$

The gradient $\partial_f l(f, y)$ can be computed in a straightforward fashion via

$$\partial_f l(f, y) = -2\frac{\sum_{y_u > y_v} C(y_u, y_v)}{m^2 - \sum_j m_j^2} \tag{7}$$

In general, computing losses using preferences such as (6) is an $O(m^2)$ operation. However, we may extend the reasoning of [12] to more than binary scores to obtain an $O(m \log m)$ algorithm instead. Algorithm 1.1 relies on sorting f before taking sums. It uses the decomposition of the soft margin loss via

$$\max(0, 1 - f_u + f_v) = \max(0, (f_v + 0.5) - (f_u - 0.5)) = \max(0, c_{v+m} - c_v)$$

where $c = [f - 0.5, f + 0.5]$ to disentangle upper and lower bounds. It then traverses the sorted list of c to check for how many terms an upper or lower bound is violated by means of auxiliary counters b and u.

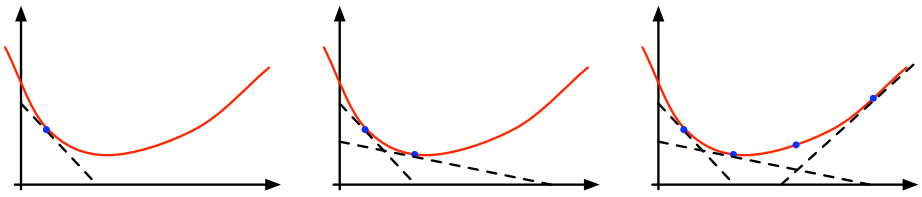

Fig. 1. A convex function (solid) is bounded from below by Taylor approximations of first order (dashed). Adding more terms improves the bound.

2.3 Optimization

Although (1) is not *jointly convex* in U and M, it is still convex in U if M is kept fixed and convex in M if U is kept fixed. We thus resort to alternating subspace descent as proposed by [10] by keeping U fixed and minimizing over M and repeating the process for M with U fixed. We have the following procedure:

repeat
  For fixed M minimize (1) with respect to U.
  For fixed U minimize (1) with respect to M.
until no more progress is made or a maximum iteration count is reached.

While this approach does not ensure that a global minimum is reached, it proves to be rather efficient and scalable for problems of up to 10^8 nonzero entries in Y (Netflix)[7].

Our implementation uses a bundle method solver for the two optimization steps over U and M. Recently, bundle methods have been introduced with promising results for optimizing regularized risk functions in supervised machine learning[8]. Bundle methods are especially suited to structured output estimation problems as they require relatively few function and gradient computations which can be costly in the structured estimation setting [13], while they can be also used to minimize non-smooth loss functions where only subdifferentials are available.

The key idea behind bundle methods is to compute successively improving linear lower bounds of an objective function through first order Taylor approximations as shown in Figure 1. Several lower bounds from previous iterations are bundled in order to gain more information on the global behavior of the function. The minimum of these lower bounds is then used as a new location where to compute the next approximation, which leads to increasingly tighter bounds and convergence.

The main computational cost in using the bundle method solver is the computation of the gradients with respect to M and U. Using the chain rule yields

$$\partial_M L(F,Y) = U^\top \partial_F L(F,Y) \text{ and } \partial_U L(F,Y) = \left[\partial_F L(F,Y)\right]^\top M. \qquad (8)$$

Algorithm 1.2. Optimization over U

input Matrix U and M, data Y
output Matrix U
for $i = 1$ **to** n **do**
  Select `idx` as the nonzero set of Y_{i*}
  Initialize $w = U_{i*}$
  $U_{i,\texttt{idx}} = \operatorname{argmin}_w l(wM_{\texttt{idx},*}, Y_{i,\texttt{idx}}) + \frac{\lambda_u}{2} \|w\|^2$
end for

Algorithm 1.3. Computation of $\partial_M L$

input Matrix U and M, data Y
output $\partial_M L = D^\top M$
for $i = 1$ **to** n **do**
  Update $w \leftarrow U_{i*}$
  Find index ind where $Y_{i*} \neq 0$
  $X \leftarrow M[ind, :]$
  Update $D_{i*} \leftarrow \partial_F L(wX, Y_{i*}[ind])$
end for$\partial_M L = D^\top M$

This computation is easily parallelizable, since terms in $\partial_F L(F, Y)$ with respect to each user can be computed separately.

We assume that the loss decomposes per user. Thus, we can optimize U by optimizing each row of U on its own as shown in algorithm 1.2. Note that we effectively construct and solve a regularized risk model for each user for a dense data matrix X and parameters w. On the other hand, when minimizing with respect to M we need to deal with the entire loss jointly. The main issue to solve is to compute the loss and its gradient with respect to M. Algorithm 1.3 shows an efficient way to compute the gradient which decomposes again for all users besides the final multiplication.

3 Extensions

So far we described a number of practical implementation details and extensions in the context of the loss function. We now discuss extensions of the model to take prior knowledge about the function class into account.

Offset Individual users may have different standards when it comes to rating movies. For instance, some users may rarely award a 5 while others are quite generous with it. Likewise, movies have an inherent quality bias. For instance, 'Plan 9 from Outer Space' will probably not garner high ratings with any movie buff while other movies may prove universally popular. This can be taken into account by means of an offset per movie. This can be incorporated via

$$F_{ij} = \langle U_{i*}, M_{j*} \rangle + u_i + m_j. \tag{9}$$

Here u and m are bias vectors for movies and users alike. In practice, we simply extend the dimensionalities of U and M by one for each bias while pinning the corresponding coordinate of the other matrix to assume the value of 1. In this form no algorithmic modification for the U and M optimization is needed. The computational cost of this extension is near zero, as the feature vectors of the convex optimization problem are extended by only one or two dimensions.

Please note that this offset is different from a simple normalization of the input data: It is learned for each user and for each movie, while a normalization cannot cater for both appropriately at the same time.

Adaptive regularization Fixing a single regularization parameter for movies and users respectively is not a very effective choice. For instance, estimation for a user who has seen very few movies will likely suffer from overfitting unless heavily regularized. On the other hand, for users who have rated lots of movies, we should be more gentle in terms of regularization. Likewise, the number of ratings per movie varies widely and the regularization should take this into account.

Those issues can be dealt with by sample-size adaptive regularization for both movies and users. Denote by D^u and D^m diagonal matrices corresponding to movies and users. Setting $D^u_{ii} = n_i^{-\alpha}$ and $D^M_{jj} = m_j^{-\alpha}$ where n_i denotes the number of movies user i has rated and m_j denotes the number of users which have rated movie j, we obtain a sample size dependent regularizer as follows:

$$\underset{U,M}{\text{minimize}}\, L(UM, Y) + \frac{\lambda_m}{2} \operatorname{tr} M^\top D^m M + \frac{\lambda_u}{2} \operatorname{tr} U^\top D^u U.$$

In our experiments we found that $\alpha = 0.5$ provides best generalization performance. This is equivalent to the regularization scales provided in a maximum a posteriori setting where the log-prior is fixed whereas the evidence scales linearly with the number of observations.

As the computation for this scaling can be done in advance, the computational cost of the adaptive regularizer is not important when compared to the overall runtime. The needed statistics can even be pre-computed and reused in many experiments.

Graph kernels So far we ignored a crucial piece of information, namely the fact that the ratings themselves are not random. For instance, knowing that a user *rated* 'Die Hard', 'Die Hard 2', and 'Top Gun' makes it likely that this user is interested in action movies. This information can be gained without even looking at the score matrix Y. One would expect that we should be able to take advantage of this structural information in addition to the actual scores.

One possibility, proposed by [9] is to use the inner product between the movies two users rent as a kernel for comparing two different users. Denote by $S_{ij} = \{C_{ij} > 0\}$. In this case they define the kernel between users i and i' to be $\langle S_{i*}, S_{i'*} \rangle$. It is well known that such a model is equivalent to using a linear model with user-features given by S. We can use this to improve the user matrix U to $U + SA$ for a suitably chosen feature matrix A.

Independently, [1] recently developed a related line of thought by assuming that the user matrix is given by $U + \bar{S}A$, where U and A are normally distributed and $\bar{S}$ is a row-normalized version of S, that is $\bar{S}_{i*} = \|S_{i*}\|_1^{-1} S_{i*}$. While their optimization strategy is very different (they use Markov Chain Monte Carlo sampling), it should already be clear at this point that the outcome is very similar to that of [9].

We now show that both approaches, which are approximately equivalent (barring the normalization of S to $\bar{S}$), are also equivalent to the use of graph kernels on the bipartite ranking graph defined between users and movies. For this purpose we require the following lemma:

Lemma 1. *Denote by $f : \mathbb{R}^n \to \mathbb{R}$ some function and let $A \in \mathbb{R}^{n \times d}$. Moreover, let $U \in \mathbb{R}^{n \times d}$ and $S \in \mathbb{R}^{n \times m}$. Then the following problems are equivalent:*

$$\underset{U,A}{\text{minimize}} \;\; f(V) + \|U\|^2 + \|A\|^2 \tag{10}$$

$$\underset{V}{\text{minimize}} \;\; f(V) + U^\top (\mathbf{1} + SS^\top)^{-1} U \tag{11}$$

Proof. Denote by (U^*, A^*) the optimal solution of (10). Clearly in this case for $V := U^* + SA^*$ the optimization problem

$$\underset{A}{\text{minimize}} \;\; f(V) + \|V - SA\|^2 + \|A\|^2 \tag{12}$$

needs to have A^* as its solution. What remains is to express A as a function of V and to show that in this case (10) and (11) are equivalent. Taking derivatives of (11) with respect to A yields

$$\partial_A \left[\|V - SA\|^2 + \|A\|^2 \right] = 2S^\top (SA - V) + 2A.$$

Hence the gradient of the objective function vanishes for $A^* = (\mathbf{1} + S^\top S)^{-1} S^\top V$. Plugging this back into (11) yields the objective function

$$\begin{aligned} & f(V) + \left\| \left[\mathbf{1} - S(\mathbf{1} + S^\top S)^{-1} S^\top \right] V \right\|^2 + \left\| (\mathbf{1} + S^\top S)^{-1} S^\top V \right\|^2 \\ = \;& f(V) + \left\| (\mathbf{1} + SS^\top)^{-1} V \right\|^2 + \left\| S^\top (\mathbf{1} + SS^\top)^{-1} V \right\|^2 \\ = \;& f(V) + V^\top (\mathbf{1} + SS^\top)^{-1} V \end{aligned}$$

Here we used the Sherman-Morrison-Woodbury identity to transform the second term in the second line. The third term follows from the fact that left and right singular vectors associated with S constitute the eigenvectors of $(\mathbf{1} + SS^\top)$ and $(\mathbf{1} + S^\top S)$ respectively. Hence we may "push" S to the left in the third term. The last equality follows by direct calculation. *

This lemma shows that the parameterization $U + SA$ (or $U + \bar{S}A$ respectively) is equivalent to using a kernel $(1 + SS^\top)^{-1}$ as regularization. The latter is well known as the inverse Laplacian kernel, since $SS^\top$ encodes the undirected graph obtained by connecting all users which watched the same movie. The connection strength in $SS^\top$ denotes the number of movies both users shared.

The net result of this reparameterization is that (10) is a computationally more efficient way of dealing with such symmetries rather than computing the inverse of $(\mathbf{1} + SS^\top)$. Since we may have millions of users the latter would be computationally infeasible.

Note also the connection to the spectral theory of graphs [14]: the eigenvalues and eigenvectors of $\mathbf{1} + \bar{S}\bar{S}^\top$ are close to those of the bipartite graph Laplacian, which can be used for clustering between movies and users respectively. This means that for similar users and movies we end up using similar parameters respectively.

4 Experiments

To evaluate the the extension described in section 3 we performed experiments on the eachmovie and movielens data sets. The data sets are summarized in table 1.

Data set	Users	Movies	Ratings
EachMovie	61265	1623	2811717
MovieLens	983	1682	100000

Table 1. Data set statistics

In order to compare the results to those in the literature, we evaluate our system using the *Normalized Discounted Cumulative Gain (NDCG)* measure:

$$DCG(Y_{i*}, \pi)@k = \sum_{j=0}^{m} \frac{2^{Y_{i\pi[j]}} - 1}{log_2(j+1)} \quad , \quad NDCG(Y_{i*}, \pi)@k = \frac{DCG(Y_{i*}, \pi)@k}{DCG(Y_{i*}, \pi_s)@k} \tag{13}$$

The permutation π is computed as the argsort of the predicted values: $\pi = argsort(F_{i*})$. The perfect permutation π_s is the argsort of the true ratings given by the user: $\pi_s = argsort(Y_{i*})$. A NDCG of 1.0 indicates that the model sorts the movies in the same order as the user. The parameter k is a cut-off beyond which the actual ranking does no longer matter. This follows the intuition that typical recommender systems can only present a limited amount of items to the user. In all our experiments, we evaluated using NDCG@10.

Following [15] and [7] , we distinguish two different evaluation scenarios: *strong* and *weak generalization*.

Weak generalization: The train set is built by sampling 10,20 or 50 movies randomly from the seen movies of each user. The ranking performance is than evaluated using the NDCG@10 score on the remaining movies.

Strong generalization: The model is evaluated on users that were not present at training time. We follow the procedure described in [15]: Movies with less than 50 ratings are discarded. The 100 users with the most rated movies are selected as the test set and the methods are trained on the remaining users. In evaluation, 10, 20 or 50 ratings from those of the 100 test users are selected. For those ratings, the user training procedure is applied to optimize U. M is kept fixed in this process to the values obtained during training. The remaining ratings are tested using the same procedure as for the weak generalization.

It is important to note that in both cases, the system is evaluated on the vast majority of movies. This mimics the true situation of a recommender system which always has more unrated items to recommend than already rated items at its disposal. However, these evaluation schemes cannot show the usefulness of the per user offset, as the number of items in the train set is fixed for all users. To evaluate the influence of this extension, we performed experiments following the weak evaluation scheme with exchanged train and test sets: The system is tested on 10, 20 or 50 items per user and trained on the remaining items.

In all experiments, the regularization parameters were fixed and not formally tuned. The dimension of U and M is fixed to $d = 10$ in all experiments. We did not observe significant performance drop when compared to a value of $d = 100$ in [7] or $d = 30$ in [1]. Additionally, we present results only for the least squares regression loss, as the emphasis is on the evaluation of the extensions to the original model. All experiments were performed ten times with different random draws of the train and test set from the data set. In total, we report results on 960 experiments.

4.1 Results

Weak generalization Table 2 shows the results for the weak generalization setting. Overall we can see that adding the offset term does improve the performance while enabling just the Graph kernel does not seem to yield significant gains in performance. This can be attributed to the fact that the impact of the graph kernel will be most profound in cases where users have rated relatively few movies compared to the average user. An additional factor to take into account is that due to computational time constraints we did not adjust the regularization parameter for each configuration. This is particularly significant in the graph kernel phase where we optimize over an additional set of parameters. Nevertheless we observe that in some cases the graph kernel seems to bring significant performance particularly when combined with the offset term. This leads us to believe that with proper parameter tuning, the graph kernel can add significant performance increases.

Inverted weak generalization The results of this set of experiments table 3, again confirm our observations in the previous setting. The offset term along often

	Method	N=10	N=20	N=50
EachMovie	Plain	0.625 ± 0.000	0.639 ± 0.000	0.641 ± 0.000
	Offset	$\mathbf{0.646 \pm 0.000}$	$\mathbf{0.653 \pm 0.000}$	$\mathbf{0.647 \pm 0.000}$
	GraphKernel	0.583 ± 0.000	0.585 ± 0.000	0.590 ± 0.001
	OffsetGK	0.576 ± 0.000	0.597 ± 0.000	0.580 ± 0.001
MovieLens	Plain	0.657 ± 0.000	0.658 ± 0.000	0.686 ± 0.000
	Offset	$\mathbf{0.678 \pm 0.000}$	0.680 ± 0.000	$\mathbf{0.701 \pm 0.000}$
	GraphKernel	0.624 ± 0.001	0.644 ± 0.000	0.682 ± 0.000
	OffsetGK	0.670 ± 0.001	$\mathbf{0.681 \pm 0.000}$	0.682 ± 0.000

Table 2. The NGDC@10 accuracy over ten runs and the standard deviation for the weak generalization evaluation.

combined with the graph kernel or the adaptive regularization brings significant performance gains. Again we have to note that the overall regularization parameters were not tuned for the individual configurations.

	Method	N=10	N=20	N=50
EachMovie	Plain	$\mathbf{0.859 \pm 0.000}$	0.731 ± 0.000	0.627 ± 0.000
	Offset	$\mathbf{0.859 \pm 0.000}$	$\mathbf{0.734 \pm 0.000}$	0.631 ± 0.000
	GraphKernel	0.837 ± 0.000	0.693 ± 0.000	0.553 ± 0.001
	AdaReg	0.858 ± 0.000	0.729 ± 0.000	0.635 ± 0.000
	AdaRegGK	0.832 ± 0.000	0.692 ± 0.000	0.578 ± 0.000
	OffsetGK	0.832 ± 0.000	0.689 ± 0.000	0.587 ± 0.001
	OffsetAdareg	$\mathbf{0.859 \pm 0.000}$	0.728 ± 0.000	$\mathbf{0.637 \pm 0.000}$
	All	0.836 ± 0.000	0.702 ± 0.000	0.585 ± 0.000
MovieLens	Plain	0.875 ± 0.000	0.750 ± 0.000	0.673 ± 0.000
	Offset	$\mathbf{0.886 \pm 0.000}$	0.764 ± 0.000	0.703 ± 0.001
	GraphKernel	0.845 ± 0.000	0.720 ± 0.001	0.667 ± 0.000
	AdaReg	0.873 ± 0.000	0.736 ± 0.000	0.652 ± 0.001
	AdaRegGK	0.835 ± 0.000	0.694 ± 0.001	0.645 ± 0.001
	OffsetGK	0.882 ± 0.000	$\mathbf{0.773 \pm 0.000}$	$\mathbf{0.703 \pm 0.000}$
	OffsetAdareg	0.874 ± 0.000	0.750 ± 0.000	0.681 ± 0.000
	All	0.869 ± 0.000	0.730 ± 0.002	0.645 ± 0.005

Table 3. The NGDC@10 accuracy over ten runs and the standard deviation for the *inverted* weak generalization evaluation.

Strong generalization For the strong generalization setting, we compare our results to Gaussian Process Ordinal Regression (GPOR) [16] Gaussian Process Regression (GPR), the collaborative extensions (CPR, CGPOR) as well as the original MMMF implementation [15], [7]. Table 4 shows our results compared to the ones from [15].

For the Movielens data, our system with the offset and graph kernel extensions outperforms the other systems we compare to. Additionally, the system

with both extensions performs consistently better than the ones with only one extension. On the Eachmovie data, our system performs the best with the offset parameters enabled. it appears that that the graph kernel in the Eachmovie data set does not improve the performance but again this can be attributed to a poor choice of the regularization parameters for this data set.

On both data sets, the results are very good in the strong generalization setting. This is not as surprising as it may seem at first: In the strong generalization phase, our system solves a convex problem for each test user in order to learn the right U_{i*} for that user. If the system learned reasonable features for the items in the weak generalization phase of this evaluation, good results for the strong evaluation phase are to be expected. We believe that this is an important benefit of our method in many applications, as it allows for fast accurate predictions for new users without the need to retrain the whole system.

	Method	N=10	N=20	N=50
EachMovie	Plain	0.615 ± 0.000	0.633 ± 0.000	0.636 ± 0.000
	Offset	$\mathbf{0.641 \pm 0.000}$	$\mathbf{0.647 \pm 0.000}$	$\mathbf{0.644 \pm 0.000}$
	GraphKernel	0.574 ± 0.000	0.581 ± 0.000	0.596 ± 0.000
	OffsetGK	0.568 ± 0.000	0.594 ± 0.000	0.579 ± 0.000
	GPR	0.4558 ± 0.015	0.4849 ± 0.0066	0.5375 ± 0.0089
	CGPR	0.5734 ± 0.014	0.5989 ± 0.0118	0.6341 ± 0.0114
	GPOR	0.3692 ± 0.002	0.3678 ± 0.0030	0.3663 ± 0.0024
	CGPOR	0.3789 ± 0.011	0.3781 ± 0.0056	0.3774 ± 0.0041
	MMMF	0.4746 ± 0.034	0.4786 ± 0.0139	0.5478 ± 0.0211
MovieLens	Plain	0.587 ± 0.001	0.644 ± 0.001	0.630 ± 0.001
	Offset	0.583 ± 0.000	0.444 ± 0.000	0.690 ± 0.000
	GraphKernel	0.613 ± 0.000	0.634 ± 0.000	0.637 ± 0.001
	OffsetGK	$\mathbf{0.684 \pm 0.000}$	$\mathbf{0.691 \pm 0.000}$	$\mathbf{0.692 \pm 0.000}$
	GPR	0.4937 ± 0.0108	0.5020 ± 0.0089	0.5088 ± 0.0141
	CGPR	0.5101 ± 0.0081	0.5249 ± 0.0073	0.5438 ± 0.0063
	GPOR	0.4988 ± 0.0035	0.5004 ± 0.0046	0.5011 ± 0.0051
	CGPOR	0.5053 ± 0.0047	0.5089 ± 0.0044	0.5049 ± 0.0035
	MMMF	0.5521 ± 0.0183	0.6133 ± 0.0180	0.6651 ± 0.0190

Table 4. The NGDC@10 accuracy over ten runs and the standard deviation for the strong generalization evaluation.

The variance over the ten runs on different data in all experiments is surprisingly low, especially given the fact that we are optimizing a non convex function. The same is true for the variance on the objective function. The low variance may mean that we always reach the same local minimum or that this minimum is indeed a global one.

Overall, the experiments show that the performance of the basic model can be significantly increased by the extensions proposed in this paper. Please note that we did not optimize the regularization parameters, which might improve

the performance even further. Additional improvements are to be expected when applying other loss functions like ordinal regression or the ranking losses as described in [7]. Both have shown to yield better ranking performance, yet we could not evaluate them here as each loss function would have added another 960 experiments.

5 Conclusion

In this paper, we have shown several extensions to the original MMMF *model* that add up to recent advances in the *optimization procedure* and the *losses*. We introduced offset terms, item dependent regularization and a graph kernel on the recommender graph. We also showed that recent extensions to MMMF [1] as well as well known approaches [9] are both instances of our graph kernel formulation.

The extensions have been introduced in a way that preserves recent extensions of MMMF to structured loss [7]. Additionally, this still allows us to use state of the art optimizers based on bundle methods which have recently been proposed for the regularized risk minimization problem[8].

On all evaluated data sets in all evaluation settings, one combination of the proposed extensions yielded significantly improved results, even though we did not tune the parameters of the model. Thus, even better results are to be expected in real world applications of this method.

The software developed to evaluate the methods described in this paper will be available on http://www.cofirank.org.

Acknowledgements

Markus Weimer has been funded under Grant 1223 by the German Science Foundation (DFG). Alexandros Karatzoglou was supported by a grant of the ANR - CADI project. We gratefully acknowledge support by the Frankfurt Center for Scientific Computing in running our experiments.

References

1. Salakhutdinov, R., Mnih, A.: Probabilistic matrix factorization. In: Advances in Neural Information Processing Systems 20, Cambridge, MA, MIT Press (2008)
2. Hoffman, T.: Latent semantic models for collaborative filtering. ACM Transactions on Information Systems (TOIS) **22**(1) (2004) 89–115
3. Srebro, N., Jaakkola, T.: Weighted low-rank approximations. In: Proceedings of the 20th International Conference on Machine Learning (ICML 2003), AAAI Press (2003) 720 – 727
4. Srebro, N., Rennie, J., Jaakkola, T.: Maximum-margin matrix factorization. In Saul, L.K., Weiss, Y., Bottou, L., eds.: Advances in Neural Information Processing Systems 17, Cambridge, MA, MIT Press (2005)
5. Srebro, N., Shraibman, A.: Rank, trace-norm and max-norm. In Auer, P., Meir, R., eds.: Proc. Annual Conf. Computational Learning Theory. Number 3559 in Lecture Notes in Artificial Intelligence, Springer-Verlag (June 2005) 545–560
6. Takács, G., Pilászy, I., Németh, B., Tikk, D.: Major components of the gravity recommendation system. SIGKDD Explorations Newsletter **9**(2) (2007) 80–83
7. Weimer, M., Karatzoglou, A., Le, Q., Smola, A.: Cofi rank - maximum margin matrix factorization for collaborative ranking. In: Advances in Neural Information Processing Systems 20, Cambridge, MA, MIT Press (2008)
8. Smola, A., Vishwanathan, S., Le, Q.: Bundle methods for machine learning. In: Advances in Neural Information Processing Systems 20, Cambridge, MA, MIT Press (2008)
9. Basilico, J., Hofmann, T.: Unifying collaborative and content-based filtering. In: Proc. Intl. Conf. Machine Learning, New York, NY, ACM Press (2004) 65–72
10. Rennie, J., Srebro, N.: Fast maximum margin matrix factoriazation for collaborative prediction. In: Proc. Intl. Conf. Machine Learning. (2005)
11. Herbrich, R., Graepel, T., Obermayer, K.: Large margin rank boundaries for ordinal regression. In Smola, A.J., Bartlett, P.L., Schölkopf, B., Schuurmans, D., eds.: Advances in Large Margin Classifiers, Cambridge, MA, MIT Press (2000) 115–132
12. Joachims, T.: Training linear SVMs in linear time. In: Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD), ACM (2006)
13. Tsochantaridis, I., Joachims, T., Hofmann, T., Altun, Y.: Large margin methods for structured and interdependent output variables. J. Mach. Learn. Res. **6** (2005) 1453–1484
14. Smola, A.J., Kondor, I.R.: Kernels and regularization on graphs. In Schölkopf, B., Warmuth, M.K., eds.: Proc. Annual Conf. Computational Learning Theory. Lecture Notes in Comput. Sci., Heidelberg, Germany, Springer-Verlag (2003) 144–158
15. Yu, S., Yu, K., Tresp, V., Kriegel, H.P.: Collaborative ordinal regression. In Cohen, W., Moore, A., eds.: Proc. Intl. Conf. Machine Learning, ACM (2006) 1089–1096
16. Chu, W., Ghahramani, Z.: Gaussian processes for ordinal regression. J. Mach. Learn. Res. **6** (2005) 1019–1041