Adaptive Collaborative Filtering

Markus Weimer
TU Darmstadt
Darmstadt, Germany
weimer@acm.org

Alexandros Karatzoglou
LITIS, INSA
Rouen, France
alexis@ci.tuwien.ac.at

Alex Smola
NICTA, Canberra 2601,
Australia
alex.smola@gmail.com

ABSTRACT

We present a flexible approach to collaborative filtering which stems from basic research results. The approach is flexible in several dimensions: We introduce an algorithm where the loss can be tailored to a particular recommender problem. This allows us to optimize the prediction quality in a way that matters for the specific recommender system. The introduced algorithm can deal with structured estimation of the predictions for one user . The most prominent outcome of this is the ability of learning to rank items along user preferences. To this end, we also present a novel algorithm to compute the ordinal loss in $O(n \log n)$ as apposed to $O(n^2)$. We extend this basic model such that it can accommodate user and item offsets as well as user and item features if they are present. The latter unifies collaborative filtering with content based filtering. We present an analysis of the algorithm which shows desirable properties in terms of privacy needs of users, parallelization of the algorithm as well as collaborative filtering as a service. We evaluate the algorithm on data provided by WikiLens. This data is a cross-domain data set as it contains ratings on items from a vast array of categories. Evaluation shows that cross-domain prediction is possible.

Categories and Subject Descriptors

H3.3 [**Information Search and Retrieval**]: Information filtering—*Collaborative Filtering*; H3.4 [**Systems and Software**]: Performance evaluation (efficiency and effectiveness); G3 [**Probability and Statistics**]: Correlation and regression analysis; G1.6 [**Optimization**]: Gradient methods—*Bundle Methods*

General Terms

Algorithms, Experimentation

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
RecSys'08, October 23–25, 2008, Lausanne, Switzerland.
Copyright 2008 ACM 978-1-60558-093-7/08/10 ...$5.00.

Keywords

Collaborative Filtering, Structured Estimation

1. INTRODUCTION

Customers of online businesses such as Netflix, Amazon or iTunes are guided through the ever increasing number of offerings via personalized recommendations. It comes as no surprise that better recommender systems are a competitive advantage in their business.

Research on better recommender systems is often targeted at specific issues which give rise to specific algorithms. For instance, collaborative filtering algorithms have the very desirable property not to depend on features of the items offered or the customers thereof. Instead, they rely on the *rating* or purchase information alone.

This is a good thing, as features are hard to get for a variety of reasons. First of all, users might be unwilling to share their data beyond some ratings or even only the purchase record. Second, many companies sell products from a vast array of categories. This makes it hard or impossible to come up with a consistent set of features to use across the catalogue. And, last but not least: Feature extraction is a laborious and error prone task.

On the other hand, features are often available and not using them would be wasteful. Thus, recent work has focused on recommender systems that combine collaborative filtering with feature based approaches [1]. In this paper, we will contribute to this track of research and also add another layer of adaptivity to collaborative filtering algorithms:

The term "better recommendations" frequently depends on the particular needs of the service providers. In many cases, a good predicted *ranking* of the items for a user along her preferences is much more desirable than a correct prediction of the actual *rating*. Recommendation systems should thus be adaptable to predict the ranking correctly at the expense of correct ratings.

Our Contribution.

We present an algorithm that stems from basic research on matrix factorizations. Careful design and analysis of this algorithm shows desirable properties:

- The algorithm does not depend on explicit features, yet it can use them if present. Thus, it is suitable for learning from multi category item sets where a common feature set is unobtainable.
- The algorithm can use a multitude of loss functions to encode the notion of a "good" recommendation. It

is set up in a way to allow any kind of notion that can be computed on the predictions for one user. One specifically interesting case is the one of ranking.

- It does not need access to the actual ratings, but only to loss values and gradients thereof (see below). Thus, the rating data can be kept decentralized. This not only reassures users, but also opens the door to another application: Collaborative Filtering as a service, where all customers benefit from each other without having to share their data.

Additionally, we present a novel, efficient algorithm to compute the ordinal regression loss and its gradient. This algorithm does so in in $O(n \log n)$ time as apposed to $O(n^2)$ of traditional approaches. We evaluate our algorithm on a new data set which poses an interesting question: Whether or not it is possible to do cross-domain collaborative filtering. The data set as well as the code run for the experiments will be made available after publishing.

The paper is organized as follows: Section 2 presents some related work on factor models for collaborative filtering. Section 3 describes the MMMF model underlying this work. We present a formulation that allows arbitrary notions of "good" recommendations to be optimized using the structured prediction framework. We also present two representative loss functions to do so. We performed experiments on a multi-category data set. These experiments are described in Section 4. We conclude the paper in Section 5 with some remarks on future extensions.

2. RELATED WORK

Recommender algorithms are traditionally divided into *Content-based* methods and *Collaborative Filtering*.

Content-based recommenders are based on methods that operate on supplied data about the users and the items. Typically, this data is in the form of text and text analysis techniques are used to extract a set of attributes characterizing an item or a user. Users and items are then matched based on this set of attributes. Content-based methods are limited by the availability and content of the data for items and users. *Collaborative filtering* recommender systems avoid the problem of data availability by analyzing the collective taste information of users. A third category of recommender systems are *hybrid* recommender systems which essentially are based on a combination of a content-based and collaborative filtering approach. Our system is mainly a collaborative filtering system but can be easily extended to a hybrid recommender system as we will describe bellow.

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 implicit or explicit loss functional [3]. The underlying idea behind these methods is that both user preferences and item properties can be modeled by a number of factors.

The known rating data in a collaborative filtering based recommender system can be thought of as 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 $Y \in \{0, \dots, 5\}^{n \times m}$, 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.

The basic idea of matrix factorization approaches is to fit the original matrix Y with a low rank approximation F. If F has rank k, user preferences are assumed to be a linear combination of only k 'model' preferences. The approximation is usually found such that it 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 (SVD) 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 [8], 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 $U \in \mathbb{R}^{n \times d}$ and $M \in \mathbb{R}^{d \times m}$ 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 infeasible in practice: the matrix norm proposed by [8] requires semi definite programming, which is feasible only for hundreds, at most, thousands of terms. Departing from this goal, Maximum Margin Matrix Factorization (MMMF) aims at minimizing the Frobenius norms of U and M, each of which is a convex problem when taken in isolation and thus tractable by current optimization techniques. It was shown in [9, 10] that optimizing the Frobenius 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 [6, 11].

Real world recommender systems often only present a few (typically ten) suggestions to the user which are ordered in terms of that user's predicted preferences. This observation is key in the reasoning of [14]. In this work, the authors propose to extend the general MMMF framework in order to minimize structured (ranking) losses instead of the sum of squared errors on the known ratings. Thus, the recommender function does no longer predict the absolute *rating* of unrated items, but the *ranking* of the top k items. To enable effective optimization of the structured ranking loss, a novel optimization technique [7] was used to minimize the loss in terms of the Normalized Discounted Cumulative Gain (NDCG).

3. STRUCTURED MAXIMUM MARGIN MATRIX FACTORIZATION

We follow the maximum margin matrix factorization approach to collaborative filtering. In matrix factorization approaches, the goal is to find matrices $U \in \mathbb{R}^{n \times d}$ and $M \in \mathbb{R}^{d \times m}$ such that $F = UM$ is close to Y. Note that F contains entries for all elements while Y only contains non zero entries for the known ratings. This approach 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. Here, a row i of U (referred to as U_{i*}) rep-

resents the feature vector for user i and a column j of M (referred to as M_{*j}) is the feature vector for item j. The predicted rating of item j by user i then is:

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

As the goal is to find the matrices U and M, the algorithm effectively learns feature vectors for both items and users.

In order to find a good factorization F into U and M, the notion of what a "good" prediction is needs to be clearly defined. Many common approaches as well as the original MMMF formulation use the notion of a squared error on the non zero entries in Y as a measure. We deviate from this by introducing *loss functions* which encode the specific quality needed in a specific recommender system.

In our system, the quality of the prediction is measured as the loss $L(F, Y)$ of having prediction F while the reality is Y. The loss ignores the non zero elements in Y, as the zeros indicate unrated items as opposed to a rating of zero. Training the system now amounts to finding U and M such that $L(F = UM, Y)$ is minimized for the known ratings. The actual loss function used is application dependent. We will discuss two appropriate loss functions for recommender systems below in Section 3.1.

However, minimizing the loss on the *known* training data will most often yield poor performance on the *unknown* data the system is applied to due to a tendency of overfitting. Overfitting is prevented in supervised machine learning by the means of a regularizer which measures model complexity. The idea is that less complex models perform better on unseen data. Example: In binary SVMs, the L_2 (Euclidian) norm of the weight vector is used as the regularization term. This corresponds to the intuition of a wide margin of separation between the two classes. For matrices, it has been shown that the Frobenius norm is an appropriate measure for complexity with a similar connotation of wide margins [10]. Hence the name *Maximum Margin* Matrix Factorization.

Minimizing both the loss and the Frobenius norm of the matrices leads to the following optimization problem:

$$\min_{U,M} L(F = UM, Y) + \frac{\lambda_m}{2}\|M\|_F^2 + \frac{\lambda_u}{2}\|U\|_F^2 \qquad (1)$$

Here λ_m, λ_u are the regularization parameters for the M and U matrix respectively. They allow for a trade-off between model complexity and prediction quality on the training data. We will discuss the actual optimization procedure of this seemingly simple objective function below in Section 3.2, after discussing the loss functions.

3.1 The Loss

We consider two main categories of loss functions. First, totally separable losses that decompose for the non-zero elements in Y. The squared loss is the most prominent example from this category. However, recommender systems typically operate in an environment where the absolute values in F don't really matter. What matters is the prediction accuracy in terms of the relative user preferences. This is reflected in the second loss category considered here which we call non separable losses. Naturally, the loss now only decomposes per user. The question "Does the system sort the elements in the same order the user would?" cannot be answered without knowing all ratings predicted for that user. To be able to use efficient optimization techniques (see below), the gradient of the loss with respect to the prediction F is of great interest.

We will now present one exemplary loss function together with its respective gradient from each category.

Squared Loss.

In the original MMMF formulation, $L(F, Y)$ was chosen to be the sum of the squared errors [9]:

$$L(F, Y) = \frac{1}{2}\sum_{i=0}^{n}\sum_{j=0}^{m} S_{ij}(F_{ij} - Y_{ij})^2 \qquad (2)$$

where

$$S_{ij} = \begin{cases} 1 & \text{if user } i \text{ rated item } j \\ 0 & \text{otherwise} \end{cases}$$

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 and 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.

To be able to use efficient optimization techniques, the gradient of the loss needs to be known. 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

$$\partial_F L(F, Y) = S.*(F - Y) \qquad (3)$$

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 items they are interested in. A fairly accurate description of which items they hate is probably less desirable. The recent paper [14] describes an optimization procedure which is capable of dealing with such problems. In general, a non separable loss takes on the following form:

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

Gradients of $L(F, Y)$ decompose immediately into per row gradients $\partial_{F_{i*}} l(F_{i*}, Y_{i*})$. 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 [2]. 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 items of score j, that is $\sum_j m_j = m$. For a given pair of items (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 an item 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 $\{f_u \leq f_v\}$ to obtain a convex differentiable loss. This yields the loss

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

The gradient of $\partial_f l(f, y)$ 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 [4] to more than binary scores to obtain an $O(m \log m)$ algorithm instead.

Algorithm 1 relies on sorting f before taking sums. It uses the decomposition of the soft margin loss via

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

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.

3.2 Optimization

The optimization problem (1) is seemingly simple, as it is unconstrained and continuous: Every choice of U and M represents a feasible, yet sometimes bad solution. However, it lacks a very desirable property to be solvable on internet scale data sets: It is not jointly convex in U and M, at least not for the loss functions discussed here. If it were jointly convex, we could apply efficient convex optimization routines to solve it.

For the original MMMF formulation using the squared loss, a direct solution can be obtained using a semi definite reformulation [9]. However, this dramatically limits the size of the problem to several thousand users / items.

Our algorithm is based on the following observation: The function (1) is, however, still convex in U and M if the other matrix is kept fixed for the losses discussed here. We can thus resort to alternating subspace descent as proposed in [5] by keeping U fixed and minimizing over M and repeating the process for M with U fixed. This leads to 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.

As to be expected when optimizing a non-convex function, this approach does not ensure that a global minimum is reached. However, it has been shown to be rather efficient and scalable at least for problems of 10^8 nonzero entries in Y (Netflix)[14].

Each optimization step is now a convex problem and thus amenable to efficient convex optimization procedures. Our implementation uses a bundle method solver. Recently, bundle methods have been introduced with promising results for optimizing regularized risk functions in supervised machine learning[7]. The bundle method used by us has been shown to need $1/\epsilon$ function evaluations to reach a solution that is ϵ-close to the optimum. Standard solvers such as LBFGS typically need $1/\epsilon^2$ evaluations.

This difference is important, as the evaluation of losses for structured output often involve computing the solution of a discrete optimization problem [12]. Example: For some ranking losses, the computation of the loss and its gradient involves the solution of a linear assignment problem in the size of the rated items of a user, which is an operation that scales with $O(n^3)$ [14].

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 be-

Algorithm 1 $(r, g) = \mathrm{l}(f, y, C)$

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 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

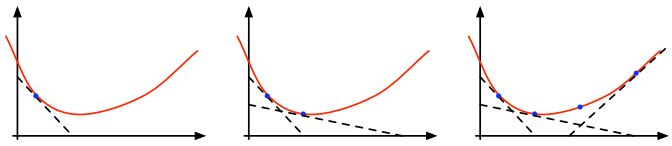

Figure 1: A convex function (solid) is bounded from below by Taylor approximations of first order (dashed). Adding more terms improves the bound.

havior 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.

We will now describe the two phases, referred to as *User Phase* and *Item Phase* in more detail.

User Phase.

The goal of the user phase is to optimize the objective function (1) with respect to U. We assume that the loss is at least of the non separable kind as introduced above. Thus, the loss and its gradient decompose per row of F and Y. Additionally, the Frobenius norm of a matrix decomposes per row, too:

$$\|U\|_F^2 = \operatorname{tr} UU^t = \sum_{i=1}^{n} \|U_{i*}\|^2$$

Thus, the optimization step over U can be decomposed into n independent optimizations, one for each user. Each of these optimizations will update a row in U:

$$\min_{U_{i*}} l(U_{i*}M, Y_{i*}) + \frac{\lambda_u}{2}\|U_{i*}\|^2 \tag{8}$$

where n denotes the number of users. Each of these n optimization problems can be solved independently.

Please note that this is a standard regularized risk minimization problem for each user besides the special treatment of the zero entries in Y. When building these optimization problems it may be advisable to compress Y_{i*} into a dense vector which not only removes this special treatment but also allows for faster computation.

As the optimization steps for the users are independent of each other, the optimization can be done in a distributed fashion.

Item phase.

The item phase is not that straightforward, as the loss does not decompose per item. If it would as in the case of the squared loss, we could resort to the very same procedure as described above for the user phase.

In the case of a non separable loss, we need to optimize the whole matrix M at once. To do so, we must be able to compute the loss as well as its gradient with respect to M. Computing the gradient of the Frobenius norm is straightforward, as the Frobenius norm decomposes per entry. We compute the gradient of $L(F = UM, Y)$ with respect to M by applying the chain rule:

$$\partial_M L(F, Y) = U^\top \partial_F L(F, Y) \tag{9}$$

Algorithm 2 Computation of $\partial_M L$

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

At first sight, this leaves us with one big computation which cannot be parallelized in any way. However, the loss and therefore its gradient with respect to F decompose per user. The actual value of the loss can be computed as the sum over this decomposition per user. We can thus compute the rows of $\partial_F L(F, Y)$ independently for each user.

The multiplication in 9 can be split into:

$$U^\top ((\partial_F L)_1 + (\partial_F L)_2 + (\partial_F L)_3 + \dots) \tag{10}$$

where $(\partial_F L)_1$ the matrix where the first row is filled with the partial gradient of the first user and the rest of the entries are zero etc.

Since the entries in the rows are sparse vectors we only need to multiply these entries with the corresponding row in the matrix U. We can thus compute $(U^\top \partial_F L)_i$ for each user i and add up the results to $\partial_M L$. Apart from decomposing an expensive dense sparse matrix multiplication we also gain in parallelization of the algorithm since we can now easily split the gradient computation onto different nodes. In practice we also observe a massive speedup when using this decomposition on a single machine.

Algorithm 2 uses this observation to compute the gradient $\partial_M L(F, Y)$.

Analysis.

The only interface of the algorithm to the actual rating data is through the loss function. All it needs for building the model is a loss value and the gradient for each iteration. Both these quantities can be computed independently for each user. This is important for several different reasons:

First of all, users need not to share their rating data with their service provider. Instead, they can just exchange losses and gradients. This may reassure privacy cautious users. Additionally, it moves one of the costliest parts of the computation off the service provider's systems and onto those of the users. Communication hardly seems like an issue, as the amount of exchanged data is dominated by the number of rated items per user.

From a similar point of view, the idea of collaborative filtering as a business to business service becomes feasible. In such a setting, an online service provider may be reluctant to share one of his key assets, the rating data. This reluctance currently leads to the problem that the quality of the predictions is mostly determined by the number of customers a service provider has. Using the algorithm described by us, service providers could keep their rating data private while still enjoying the benefit of a way better estimation of M.

Finally, these properties make it trivial to parallelize the algorithm onto a cluster of compute nodes. There, each node would be responsible for a certain number of users and computes the loss and the gradient thereof for these users. This allows the algorithm to be run with loss functions that would otherwise be prohibitively expensive.

3.3 Extensions

After presenting MMMF as well as its formulation for structured estimation and efficient training procedures to do so, we will now discuss extensions to this model to accommodate user and item offsets as well as their known features.

User and Item Offsets.

Individual users may have different standards when it comes to rating items. For instance, some users may rarely award a 5 while others are quite generous with it. This behavior can be modeled using an offset term for each user.

Likewise, items have an inherent quality bias. For instance, the movie '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 item.

Both offsets can be incorporated into the prediction via the following extension:

$$F_{ij} = \langle U_{i*}, M_{*j} \rangle + u_i + m_j. \tag{11}$$

Here $u \in \mathbb{R}^n$ and $m \in \mathbb{R}^m$ are bias vectors for items 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.

User and Item Features.

The approach as we discussed it until now does not need features. However, they might be present in some applications and it is advisable to make use of them if so. The way to introduce item features is by optimizing an additional weight vector for these for each user:

$$F_{ij} = \langle U_{i*}, M_{*j} \rangle + \left\langle W^U_{i*}, X^M_{j*} \right\rangle$$

Here, $W^U \in \mathbb{R}^{n \times d_M}$ is a matrix whose rows are the weight vectors for the item features per user. The matrix $X^M \in \mathbb{R}^{m \times d_M}$ consists of the feature vectors of the items as its rows. The number of features known for the items is d_M.

The very same idea can be applied to user features as well. The intuition here is that e.g. a certain movie might be preferred by a certain demographic group. Demographic information may be present as user features and included into the model using the very same procedure:

$$F_{ij} = \langle U_{i*}, M_{*j} \rangle + \left\langle W^U_{i*}, X^M_{j*} \right\rangle + \left\langle W^M_{i*}, X^U_{j*} \right\rangle \tag{12}$$

The matrices $W^M \in \mathbb{R}^{m \times d_U}$ and $X^U \in \mathbb{R}^{n \times d_U}$ encode the weight vector for each item and the feature vector for each user. The number of user features is d_U.

As with the offsets, the features can be integrated into the algorithmic procedure described above without any changes. To do so, one would extend M with the features from X^M, U with the features from X^U. These new entries are masked from optimization and regularization such that their value stays fixed. To learn the parameters, U is extended by W^U and M by W^M. The optimization over U and M includes these new entries. This yields the parameter vectors W^U and W^M. Please note that the Frobenius norm decomposes per entry in U and M. Thus, the newly introduced parameters are regularized as if there would be a L_2 norm imposed on them.

4. EXPERIMENTS

The algorithm presented does not need explicit features. Thus, it can be applied to domains where the items rated stem from very different categories where a common set of features is hard or impossible to obtain. Most commonly used evaluation data sets focus on one item domain only, mostly movies. Similar MMMF based systems have been evaluated on these data sets such as Movielens, Eachmovie and Netflix with promising results [6, 11, 14]. To the best of our knowledge, the MMMF idea has not been evaluated on multi domain data sets. This is probably due to the fact that those data sets are not commonly available.

We evaluated the system on data kindly provided by `wikilens.org`. WikiLens is a website (a wiki) where people can rate items on a five star scale just as on other commercial websites such as Netflix. What sets WikiLens apart is the process in which items are selected to be rate-able in the first place: In web shops, the items are purchasable from the site and the rate-able items thus form the catalogue of the service provider. At WikiLens, the items as well as their categorization are edited in a community effort. Every user of the site can add new items or categories thereof to the system. Thus, the data set contains ratings on a diverse set of items from a diverse set of categories such as "Activity" or "Beer". The data set is rather small by web standards: It consists of $26,937$ ratings by 326 users on $5,111$ items from 36 categories.

To measure the rating accuracy of the system, we resorted to the well established Root Mean Squared Error (RMSE) measure:

$$RMSE(F, Y) = \sqrt{\frac{\sum_{i=0}^{n} \sum_{j=0}^{m} S_{ij}(F_{ij} - Y_{ij})^2}{\sum_{i=0}^{n} \sum_{j=0}^{m} S_{ij}}} \tag{13}$$

where

$$S_{ij} = \begin{cases} 1 & \text{if user } i \text{ rated item } j \\ 0 & \text{otherwise} \end{cases}$$

Table 1 shows the results for this evaluation. All Experiments where run ten times on ten independent samples of test and training data. We used 90% of the ratings of a user as training data and 10% as test data. We fixed d, the dimensionality of U and M to the value of 10. This may seem very low, also compared to results reported on other data sets where it was set to 30 [6] or even 100 [14]. We did some experiments with higher values of d, and those experiments did not show very different results. This is interesting in its own right.

We ran all experiments with both the squared error loss (labelled with Regression in the tables) as well as the ordinal loss (Ordinal). We performed a very coarse parameter

search on λ_U and λ_M for the possible values $0.1, 1.0$ and 10.0. For the squared loss, we performed a very simple yet effective preprocessing of the data: From all entries of Y, we subtracted the average rating. Note that the RMSE scores are comparable to the others reported in Table 1:

$$\begin{aligned} RMSE(\hat{F}, \hat{Y}) &= \sqrt{\frac{\sum_{i=0}^{n}\sum_{j=0}^{m} S_{ij}((F_{ij}-a)-(Y_{ij}-a))^2}{\sum_{i=0}^{n}\sum_{j=0}^{m} S_{ij}}} \\ &= \sqrt{\frac{\sum_{i=0}^{n}\sum_{j=0}^{m} S_{ij}(F_{ij}-Y_{ij})^2}{\sum_{i=0}^{n}\sum_{j=0}^{m} S_{ij}}} \\ &= RMSE(F, Y) \end{aligned}$$

Here, $\hat{F}, \hat{Y}$ are the mean corrected variants of F, Y and a is the mean of the non zero entries in Y.

We observe in Table 1, that the regression loss outperforms the ordinal loss in our experiments in all instances. This is to be expected, as the ordinal loss does not optimize for correct prediction of the ratings but for a correct ordering of the items.

Additionally, the ordinal loss gains more from the offsets. This is also to be expected, as the preprocessing and the offsets have a similar yet not completely identical effect. A small test of the ordinal loss on the preprocessed data showed that it's results would be far worse with this preprocessing, in the order of an RMSE score of 3.

A value of 10 for λ is very commonly the one yielding the best results. This suggests that with a more elaborate parameter tuning around this value, the results should be enhanced.

The absolute value of the RMSE is below average when compared to known results on huge single-domain data sets (the Netflix baseline is 0.96). As there are no published results of other approaches on this data set, we computed a simple baseline to compare to: Given the train data, use its mean as the constant prediction. This procedure yields a RMSE score of 1.15, which is significantly worse than our best system configuration. It is plausible that the score of our system can be further improved by sophisticated preprocessing and parameter tuning techniques which are beyond the scope of this paper. For the purpose of this paper, the results do show that prediction in multi-domain data sets can be done using MMMF-style algorithms.

In many real world recommenders, the absolute value of the predicted ratings is of less interest than the correct prediction of relative preferences for a user. To measure this *ranking* performance of the algorithm, we used the *Normalized Discounted Cumulative Gain (NDCG)* as described in [13]:

$$DCG(Y_{i*}, \pi)@k = \sum_{j=0}^{m} \frac{2^{Y_{i\pi[j]}} - 1}{log_2(j+1)} \quad (14)$$

$$NDCG(Y_{i*}, \pi)@k = \frac{DCG(Y_{i*}, \pi)@k}{DCG(Y_{i*}, \pi_s)@k} \quad (15)$$

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. NDCG puts an emphasis on getting the first ratings right. See Figure 2 for an example of the DCG scores obtained when exchanging the first and when exchanging the last element of the ranking with the middle one.

Perfect permutation of ratings	★★★★★	★★★★	★★★	★★	★	Σ
Contribution to DCG	44.72	13.65	5.05	1.86	0.56	**65.84**
First wrong	★★★	★★★★	★★★★★	★★	★	Σ
Contribution to DCG	10.10	13.65	22.36	1.86	0.56	**48.53**
Last wrong	★★★★★	★★★★	★	★★	★★★	Σ
Contribution to DCG	44.72	13.65	0.72	1.86	3.91	**64.85**

Figure 2: DCG scores for different orderings.

Many recommender systems can only present a limited amount of recommendations to their users, typically in the order of ten. Thus, the performance on items which are never presented to the user is neglect-able. This reasoning leads to the introduction of the cut-off parameter k, beyond which the actual ranking does no longer matter. In all our experiments, we evaluated using NDCG@10.

The experiments where run in the very same fashion as for RMSE: All Experiments where run ten times on ten independent samples of test and training data. We used 90% of the ratings of a user as training data and 10% as test data. We performed a very coarse parameter search on λ_U and λ_M for the possible values $0.1, 1.0$ and 10.0.

Table 2 shows the results for this evaluation. Surprisingly, the regression loss performs better at the ranking task than the ordinal loss. To some extend, this is due to the nature of the offsets. They have at least the impact of a normalization of the ratings per item and per user. Thus, the task of minimizing the squared distance and the relative ranking become very similar.

All experiments yield very high NDCG scores when compared to those reported for other data sets which usually are around 0.7. This is mostly due to the fact that the data set is rather small: Picking and ordering the "right 10" items from a catalogue of tens of thousands of items is way harder than what we can evaluate on this data set: The average number of ratings by a user is 82. Thus, the average evaluation is done on less than 10 items, as 10% of the ratings for each user form the test set. Thus, there is no penalty any more for having the wrong elements in the top 10 of the ranking.

In all our experiments, the variance over ten runs on different data samples is very low, often smaller than 0.001. This indicates that the method is very stable with respect to noise in the data. Even if our method does not guarantee global convergence, it seems to converge to the same local minima every time. This may or may not be the global minimum, though.

5. CONCLUSION

In this paper, we presented several extensions to maximum margin matrix factorization. First, the usage of arbitrary loss functions which paves the way to structured prediction. Inside this framework, we presented a novel and efficient algorithm for the optimization of the ordinal ranking loss. We extended the general MMMF framework with item and movie offsets as well as features.

	User-Offset	Item-Offset	RMSE	λ_U	λ_M
Regression	No	No	1.12 ± 0.00	10.0	10.0
	No	Yes	1.10 ± 0.00	10.0	10.0
	Yes	No	1.09 ± 0.00	10.0	10.0
	Yes	Yes	1.10 ± 0.00	10.0	10.0
Ordinal	No	No	1.63 ± 0.00	10.0	10.0
	No	Yes	1.44 ± 0.00	10.0	10.0
	Yes	No	1.42 ± 0.00	0.1	0.1
	Yes	Yes	1.40 ± 0.00	10.0	10.0

Table 1: The RMSE performance over ten runs and the variance for different system configurations. Lower is better.

	User-Offset	Item-Offset	NDCG	λ_U	λ_M
Regression	No	No	0.882 ± 0.000	10.0	10.0
	No	Yes	0.883 ± 0.000	10.0	10.0
	Yes	No	0.887 ± 0.000	10.0	10.0
	Yes	Yes	0.888 ± 0.000	10.0	10.0
Ordinal	No	No	0.879 ± 0.000	10.0	10.0
	No	Yes	0.884 ± 0.000	10.0	10.0
	Yes	No	0.880 ± 0.000	10.0	10.0
	Yes	Yes	0.882 ± 0.000	10.0	1.0

Table 2: The NDCG@10 accuracy over ten runs and the variance for different system configurations. Higher is better.

To the best of our knowledge, we report the first results of a MMMF-style algorithm on the WikiLens data set. This data set consists of ratings on many different item categories, which sets it apart from classical data sets such as Movielens, Netflix and Eachmovie. Our results indicate that the introduced extensions are vital for the performance on this and most probably similar data sets. The results are promising and suggest even better results on bigger data sets.

Analysis of the algorithm showed that it can be implemented in a privacy-cautious way in the sense that the actual ratings never need to be presented to the algorithm, only losses and gradients. This opens the door both to data from privacy concerned users as well as collaborative filtering as a service, where the owner of the rating data does not need to share it with the provider of the collaborative filtering service. Yet, all customers of the collaborative filtering service can still benefit from each other. We will investigate the feasibility of this application in the future.

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.

6. REFERENCES

[1] R. Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, 2002.

[2] R. Herbrich, T. Graepel, and K. Obermayer. Large margin rank boundaries for ordinal regression. In A. J. Smola, P. L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 115–132, Cambridge, MA, 2000. MIT Press.

[3] T. Hoffman. Latent semantic models for collaborative filtering. *ACM Transactions on Information Systems (TOIS)*, 22(1):89–115, 2004.

[4] T. Joachims. Training linear SVMs in linear time. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 2006.

[5] J. Rennie and N. Srebro. Fast maximum margin matrix factoriazation for collaborative prediction. In *Proc. Intl. Conf. Machine Learning*, 2005.

[6] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems 20*, Cambridge, MA, 2008. MIT Press.

[7] A. Smola, S. Vishwanathan, and Q. Le. Bundle methods for machine learning. In *Advances in Neural Information Processing Systems 20*, Cambridge, MA, 2008. MIT Press.

[8] N. Srebro and T. Jaakkola. Weighted low-rank approximations. In *Proceedings of the 20th International Conference on Machine Learning (ICML 2003)*, pages 720 – 727. AAAI Press, 2003.

[9] N. Srebro, J. Rennie, and T. Jaakkola. Maximum-margin matrix factorization. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, Cambridge, MA, 2005. MIT Press.

[10] N. Srebro and A. Shraibman. Rank, trace-norm and max-norm. In P. Auer and R. Meir, editors, *Proc. Annual Conf. Computational Learning Theory*, number 3559 in Lecture Notes in Artificial Intelligence, pages 545–560. Springer-Verlag, June 2005.

[11] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Major components of the gravity recommendation system. *SIGKDD Explorations Newsletter*, 9(2):80–83, 2007.

[12] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *J. Mach. Learn. Res.*, 6:1453–1484, 2005.

[13] E. Voorhees. Overview of the trect 2001 question answering track. In *TREC*, 2001.

[14] M. Weimer, A. Karatzoglou, Q. Le, and A. Smola. Cofi rank - maximum margin matrix factorization for collaborative ranking. In *Advances in Neural Information Processing Systems 20*, Cambridge, MA, 2008. MIT Press.