Mind the Gaps: Weighting the Unknown in Large-Scale One-Class Collaborative Filtering

Rong Pan
HP Labs,
1501 Page Mill Rd
Palo Alto, CA 94304, USA
rong.pan@hp.com

Martin Scholz
HP Labs,
1501 Page Mill Rd
Palo Alto, CA 94304, USA
scholz@hp.com

ABSTRACT

One-Class Collaborative Filtering (OCCF) is a task that naturally emerges in recommender system settings. Typical characteristics include: Only positive examples can be observed, classes are highly imbalanced, and the vast majority of data points are missing. The idea of introducing weights for missing parts of a matrix has recently been shown to help in OCCF. While existing weighting approaches mitigate the first two problems above, a sparsity preserving solution that would allow to efficiently utilize data sets with e.g., hundred thousands of users and items has not yet been reported. In this paper, we study three different collaborative filtering frameworks: Low-rank matrix approximation, probabilistic latent semantic analysis, and maximum-margin matrix factorization. We propose two novel algorithms for large-scale OCCF that allow to weight the unknowns. Our experimental results demonstrate their effectiveness and efficiency on different problems, including the Netflix Prize data.

Categories and Subject Descriptors

G.1.2 [**Approximation**]: Least squares approximation; H.2.8 [**Database Applications**]: Data Mining

General Terms

Algorithms

Keywords

Collaborative Filtering, Large-Scale, One-Class

1. INTRODUCTION

Collaborative filtering is at the core of modern personalization machinery. The longer-term vision is to be able to automatically stream the right kind of news to our desks and to recommend the movies and music we have not yet noticed we desired, without ever having to bother with search queries. Recommender systems have found their way into practice over the last decade. Prominent examples include Amazon's product recommender and the Google News website. Those systems leverage a large pool of training observations and rely to a large part on implicit feedback, i.e., which user performed which action.

Our work addresses a particularly hard case that emerges from several real-world prediction tasks: We try to learn from implicit feedback only, and under the restriction that each observation is a positive example of e.g., a user's interest. Examples include (i) tracking which items a user bought in the past, and trying to predict the current interests, or (ii) using the (social) bookmarks of a user to predict which other yet unknown sites she also might like. In both these cases, the observed events are positive examples of what a user likes, so all observations belong to a single class.

There are two different strategies for handling such one-class problems. The first one is to assume that all unobserved events (e.g., sites that were not bookmarked) are negative examples. This is obviously simplistic, but allows one to restate the problem in terms of traditional binary classification. We refer to this strategy as *all missing as negative* (AMAN). The second strategy is to explicitly state that the missing values are unknown, and to turn to probabilistic density estimation techniques that are capable of working with positives only. We refer to this strategy as *all missing as unknown* (AMAU). We recently proposed the use of instance weighting techniques after substituting zeros for unknown values [13]. This allows us to balance the extreme cases of AMAN and AMAU: A weight of one after substitution is identical with the AMAN strategy, whereas a weight of zero results in an AMAU strategy. Choosing fractional weights reflects degrees of confidence that missing values are in fact negatives.

This paper entails an empirical study that covers the major families of algorithms: an AMAU-based probabilistic model, an AMAN technique based on a low-rank approximation, and another AMAN technique that is based on a maximum margin approach. Our previous empirical study was limited by the scalability of algorithms [13]. Since collaborative filtering systems benefit from large user bases, we mostly focus on scalability aspects and the performance on large-scale benchmarks in this paper. Our goal is to gain a better understanding which methods are truly valuable in practice, and how much we gain by introducing weights. As a major contribution, we propose two scalable algorithms that allow to weight missing values in sparse matrices. The first one is based on low-rank approximations, the second one extends maximum-margin matrix factorization.

2. LARGE-SCALE OCCF STRATEGIES

2.1 Formal Background

The common goal of techniques discussed in this paper is to decompose a given $n \times m$ matrix $\boldsymbol{R}$, for example describing which news articles were read by which users. We investigate the predictive performance of three different families or tools. The first one (pLSI) is based on a probabilistic model, the second one (ALS) on low-rank approximations, and the third one follows the paradigm of maximum-margin classification. Surprisingly, despite those deep semantic differences, all three methods yield a very similar output: All models consist of two matrices $\boldsymbol{X} = (X)_{n \times d}$ and $\boldsymbol{Y} = (Y)_{m \times d}$ that are chosen so that $\boldsymbol{X}\boldsymbol{Y}^T$ approximates the original matrix of observations $\boldsymbol{R}$. We will discuss the different techniques in more depth in the following sections.

Regarding notation, we will use bold upper-case letters for matrices and vectors throughout this work. A matrix with a single index e.g., $\boldsymbol{R}_r$ denotes the r-th row vector. Individual components are referred to when using two indices, e.g., $R_{r,c}$ denotes the element in row r and column c. For column vectors we use the notation $\boldsymbol{R}_{\cdot,c}$. For any matrix $\boldsymbol{R}$, $||\boldsymbol{R}||^2_F$ denotes the Frobenius norm. The vector $\mathbf{1}$ denotes the column vector that has 1 as the value of each component. Its dimensionality can be concluded from the context. Finally, $\mathbf{I}$ refers to the identity matrix.

Let $\boldsymbol{R} = (R)_{n \times m}$ be a user-item matrix. As discussed in the problem statement, $\boldsymbol{R}$ contains only positives, since only positives are observable. We might represent those as entries with a value of 1. Unobserved values are not necessarily negative though, so we try to avoid respresenting them using any fixed value at this point. Let instead $\mathcal{M} := \{1, \ldots, m\}$ be the set of all column indices of $\boldsymbol{R}$, that is, the set of all items, and $\mathcal{M}_r$ denote the set of indices of all *observed* (hence positive) items in row $\boldsymbol{R}_r$.

2.2 Probabilistic Latent Semantic Indexing

The first model we want to include in our study is called probabilistic latent semantic indexing (pLSI) [9]. It was used in the context of the Google News recommender engine [7], an application that is very close to OCCF.

PLSI is based on a generative model: Each user has a distribution over interests, and each interest "generates" the action of picking a particular item with a fixed probability. Interests are modeled as latent topics z that connect users and items in a graphical model. The model specifies conditional probabilities $P(z|u)$ for each user u to like / pick each of the latent topics z, and for each topic z to generate a particular item i, $P(i|z)$. The assumption is that the choice of an item is independent of the user given the latent topic. The probability that user u picks item i simplifies to

$$P(u, i) = \sum_z P(i|z)P(z|u)$$

under this assumption. Since pLSI follows a probabilistic framework, the goal is to find a maximum likelihood model that best explains the observations. There is a well-known EM algorithm for this task [9].

The pLSI model and algorithm provide a good example of an AMAU strategy, because the model operates exclusively on the positives. The resulting estimates of $P(u, i)$ can be used to rank items for each user according to the estimated likelihoods of preference for each item.

```
Require: data matrix $\boldsymbol{R} \in \mathbb{R}^{n \times m}$, rank $d$
Ensure: Matrices $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ and $\boldsymbol{Y} \in \mathbb{R}^{m \times d}$
  Initialize $\boldsymbol{Y}$ randomly
  repeat
    Update $\boldsymbol{X}_r, \forall r \in \{1, \ldots, n\}$
    Update $\boldsymbol{Y}_c, \forall c \in \{1, \ldots, m\}$
  until convergence.
  return $\boldsymbol{X}$ and $\boldsymbol{Y}$
```

Algorithm 1: Alternating Least Squares (ALS)

Given the AMAU nature of pLSI, introducing weights into this framework is only reasonable for non-missing values. As we will discuss later, we are exclusively looking into the problem of weighting the missing values in this paper, so we skip the option of weighting pLSI.

2.3 Least Squares Matrix Decomposition

2.3.1 *Regular Alternating Least Squares*

The techniques discussed throughout Section 2.3 have their roots in singular value decomposition (SVD). For a given matrix $\boldsymbol{R} \in \mathbb{R}^{n \times m}$, an SVD is a decomposition of the matrix $\boldsymbol{R} = \boldsymbol{X}\mathbf{S}\boldsymbol{Y}^T$, where $\boldsymbol{X} \in \mathbb{R}^{n \times r}$ and $\boldsymbol{Y} \in \mathbb{R}^{m \times r}$ are orthonormal matrices, r is the rank of $\boldsymbol{R}$, and $\mathbf{S}$ is a diagonal matrix with the singular values of $\boldsymbol{R}$ on its main diagonal. When projecting $\boldsymbol{X}$, $\mathbf{S}$, and $\boldsymbol{Y}$ onto the $d \leq r$ columns for which S has the highest singular values, the product of the three resulting matrices gives the best approximation $\tilde{\boldsymbol{R}}$ of $\boldsymbol{R}$ of rank d with respect to $||\boldsymbol{R} - \tilde{\boldsymbol{R}}||^2_F$.

Unfortunately, the SVD framework loses its convexity if it encounters missing values. The alternating least squares algorithm (ALS) provides an efficient search strategy and handles missing values gracefully, but only guarantees to find local optima in such cases. Due to its success in the Netflix Prize (where most of the values are missing) ALS gained a lot of attention recently. It fits the objective of this competition well: to optimize for the Frobenius norm.

The variations of ALS discussed in this work all have the skeleton depicted in Algorithm 1 in common, and only differ in terms of the loss function and corresponding updates of $\boldsymbol{X}$ and $\boldsymbol{Y}$. The algorithm starts with a random matrix $\boldsymbol{Y}$, and then alternates steps of optimizing $\boldsymbol{X}$ for fixed $\boldsymbol{Y}$ and of optimizing $\boldsymbol{Y}$ for fixed $\boldsymbol{X}$. Since both these steps are perfectly symmetric, we will for notational simplicity just discuss the case of optimizing $\boldsymbol{X}$ throughout this section.

Some additional notation helps describing those updates. ALS exploits sparsity and handles missing values by projecting matrices and vectors so that missing values in $\boldsymbol{R}$ have no effect. Let π_r denote a function that projects exactly those components of a column vector into a lower dimensional space for which the indices are not missing in the vector $\boldsymbol{R}^T_r$, that is, it projects the components with index in $\mathcal{M}_r$ into a $|\mathcal{M}_r|$-dimensional subspace. Analogously, for a matrix $\boldsymbol{Y}$, let $\pi_r(\boldsymbol{Y})$ denote the matrix that results from projecting each column vector $\boldsymbol{Y}_{\cdot,c}$ to $\pi_r(\boldsymbol{Y}_{\cdot,c})$. For illustration, if no values are missing in $\boldsymbol{R}_r$, then

$$\pi_r(\boldsymbol{Y})^T \pi_r(\boldsymbol{R}^T_r) = \boldsymbol{Y}^T \boldsymbol{R}^T_r,$$

otherwise the multiplication after projection (left hand side) simply ignores all products containing a missing value.

We use ALS with Tikhonov-regularization (parameter λ) in this work, as regularization consistently gives better results. We refer to it as "regular ALS". Its loss function is

$$\mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}) = ||\boldsymbol{R} - \boldsymbol{X}\boldsymbol{Y}^T||_F^2 + \lambda \left(||\boldsymbol{X}||_F^2 + ||\boldsymbol{Y}||_F^2\right).$$

When updating $\boldsymbol{X}$ for given $\boldsymbol{R}$, $\boldsymbol{Y}$, and λ, each row $\boldsymbol{X}_r$ of $\boldsymbol{X}$ can be updated separately, and there is a closed-form solution for minimizing the loss:

$$\boldsymbol{X}_r := \left(\pi_r(\boldsymbol{Y})^T \pi_r(\boldsymbol{Y}) + \lambda \mathbf{I}\right)^{-1} \pi_r(\boldsymbol{Y})^T \pi_r(\boldsymbol{R}_r^T) \tag{1}$$

Besides having this neat closed form solution, ALS has a few other pleasant properties worth mentioning, see also [21]: First, all ALS variants we evaluated had a stable behavior when it came to parameter tuning. We found it to be substantially less sensitive to parameter settings and more stable in its performance than e.g., pLSI. Second, the ALS variant that includes regularization hardly ever overfits, so by increasing the number of latent features d (the rank of $\tilde{\boldsymbol{R}}$) ALS tends to monotonically improve its predictive performance, just with diminishing returns for every extra CPU cycle (additional features). Finally, it is easy to parallelize.

2.3.2 Prior work on ALS for OCCF

For one-class problems, the ALS framework seems a bit unmotivated at first. ALS provides a solution to a regression-style matrix completion problem, but in OCCF we only have one kind of observed value, and all the remaining values are missing. However, as a first step we can use the AMAN substitution and rephrase OCCF as a binary classification-style problem, which is a special case of regression with the response being either 0 or 1. A regularized low-rank approximation will fail to reproduce the matrix exactly, so for each user we can rank items with originally unknown values with respect to the predicted values in $\tilde{\boldsymbol{R}}$.

The results in [13] indicate that this approach, which we refer to as *ALS after AMAN substitution*, benefits from introducing weights for the originally unknown values of $\boldsymbol{R}$; different weighting schemes based on whether a value is present or missing, and – optionally – based on the individual user and item under consideration improved the predictive power of collaborative filtering models.

A straightforward adaptation of the loss function $\mathcal{L}$ above supports this kind of weighting [16]: If $\boldsymbol{W}$ is the weight matrix, then we define *weighted loss with respect to* $\boldsymbol{W}$ as

$$\begin{aligned}\mathcal{L}_W(\boldsymbol{X}, \boldsymbol{Y}) \;&:=\; \sum_{i,j} W_{i,j} \Big((R_{i,j} - X_i Y_j^T)^2 \\ &+\; \lambda(||\boldsymbol{X}_i||_F^2 + ||\boldsymbol{Y}_j||_F^2) \Big)\end{aligned}$$

The problem of updating $\boldsymbol{X}$ to minimize the loss $\mathcal{L}_W$ still has a closed-form solution after AMAN substitution [13]:

$$\boldsymbol{X}_r := \boldsymbol{R}_r \widetilde{\boldsymbol{W}_r} \boldsymbol{Y} \left(\boldsymbol{Y}^T \widetilde{\boldsymbol{W}_r} \boldsymbol{Y} + \lambda (\sum_{i \in \mathcal{M}} W_{r,i}) \mathbf{I} \right)^{-1}, \tag{2}$$

where $\widetilde{\boldsymbol{W}_r}$ is a $n \times n$ diagonal matrix with the weights of row r on the main diagonal. In our experiments, we will only consider the three weighting schemes proposed in [13].

The basic idea in all three cases is to let $\boldsymbol{W}$ reflect the credibility of the training data ($\boldsymbol{R}$) that we use to build a collaborative filtering model. Positive examples are likely indeed positive. Given that the information of this kind is rare in typical one-class applications, we trust the data in this case, and set $W_{ij} = 1$ whenever $R_{ij} = 1$.

Table 1: Weighting Schemes

	Pos Examples	"Neg" Examples
Uniform	$W_{i,j} = 1$	$W_{i,j} = \delta$
User-Oriented	$W_{i,j} = 1$	$W_{i,j} \propto \sum_j R_{i,j}$
Item-Oriented	$W_{i,j} = 1$	$W_{i,j} \propto m - \sum_i R_{i,j}$

In turn, missing data points are likely to be negative examples. In social bookmarking, for example, each user has very few web pages and tags; for news recommendation, each user reads only a small fraction of the available articles. We observe that the confidence in missing values being negative is not as high as in non-missing values being positive, which suggests to assign lower weights to allegedly negative examples. This is addressed by our first weighting scheme. It uniformly assigns a confidence weight $\delta \in [0, 1]$ to each "negative" example. The rationale behind the second weighting scheme is that the available data on some users is poor, so we should lower our confidence in the data for these users and not trust missing parts to be negative. Poor information means that we have only a few observations (positive examples) for a user. The third weighting scheme resembles the popularity-based ranking of items. Here we expect to have more positives in the missing parts if the items are more popular. Table 1 summarizes these three schemes. We plan to investigate more complex weighting schemes in our future work.

There are two reasons why we do not consider weighting the non-missing values in this paper. First, the success of weighting known values inherently depends on the quality and encoding of background knowledge into weights (e.g., [20]); in this case, we do not even trust our very few *known* data points after all. Collecting and incorporating background knowledge is an orthogonal problem, that is out of the scope of this paper. Second, weighting just the non-missing values for all three matrix decomposition techniques, PLSA, ALS, and MMMF, does not increase asymptotic runtime complexity. It is straightforward to extend all subsequently proposed techniques in this fashion.

2.3.3 Runtime analysis

Scalability is decisive for practical OCCF applications. We first look into the complexity of regular ALS. Let in the subsequent analysis $|\boldsymbol{R}|$ denote the number of non-missing values in $\boldsymbol{R}$. Regular ALS can basically "handle" missing values by ignoring them. When updating the r-th row of $\boldsymbol{X}$ according to Eq. (1), ALS only requires the projection $\pi_r(\boldsymbol{Y})$, or equivalently, only the columns of $\boldsymbol{Y}^T$ for which the column in the r-th row of $\boldsymbol{R}$ is not missing. In total, the whole recomputation of $\boldsymbol{X}$ (for all rows together) hence requires to read $|\boldsymbol{R}|$ rows from $\boldsymbol{Y}$. In the next step, $\pi_r(\boldsymbol{Y})^T \pi_r(\boldsymbol{Y}) + \lambda \mathbf{I}$ can trivially be computed in time $O(d^2 \cdot |\mathcal{M}_r|)$ for each individual row r. This term dominates the previous read operation and the subsequent multiplications with $\pi_r(\boldsymbol{Y})^T$ and $\pi_r(\boldsymbol{R}_r^T)$. In practice, it will usually also dominate the n matrix inversions, although for very sparse matrices or large d one should keep in mind that the costs per matrix inversion are cubic when using naive implementations, and still no better than $O(d^{2.376})$ even for the fastest known algorithm [6]. The total costs for updating $\boldsymbol{X}$ (or analogously $\boldsymbol{Y}$) without the matrix inversions are in $O(|\boldsymbol{R}| \cdot d^2)$, since the sum of $|\mathcal{M}_r|$ over all rows is $|\boldsymbol{R}|$. Putting together all costs and assuming a naive matrix inversion we still end up with an upper bound of only

$$O\left(|\boldsymbol{R}| \cdot d^2 + (n+m) \cdot d^3\right) \tag{3}$$

per iteration when updating both $\boldsymbol{X}$ and $\boldsymbol{Y}$ with regular ALS. For a fixed number of latent variables d, computational costs scale linearly with the number of non-missing values.

Regular ALS lacks the good generalization performance of its gap-weighting counter-part though [13]. The weighting algorithm proposed in [20] is somewhere in between. It supports weights only for the small set of non-missing values, which does not increase the runtime complexity above. In order to support weights for missing values we want to apply methods as described in [13] (see Eq. (2)). Unfortunately, explicitly substituting all missing values with zeros and weighting the full matrix increases the runtime complexity from $O(|\boldsymbol{R}|)$ to $\Omega(n \cdot m)$. Collaborative filtering relies on a large number of users and items and is performed on sparse matrices, so this last strategy is impractical.

One work-around that avoids this strategy is an ensemble technique that runs ALS multiple times [13]. Each iteration uses only a subsample of negative examples, while ignoring the (still) missing values just like regular ALS does. This makes the approach feasible in practice, but (i) it decreases the amount of negative examples considered during training, which reduces the expected predictiveness of results, and (ii) it still increases the runtime considerably compared to the case of ALS without substituting any examples, since ALS is run multiple times on expanded data sets.

2.3.4 A novel gap-weighting algorithm that scales

We next describe an algorithm that is capable of solving the above-mentioned ALS optimization problem with weighted missing values very efficiently. It supports all weighting schemes shown in Table 1 at the *same* asymptotic computational costs as the regular ALS algorithm, see Eq. (3). In fact, the increase in runtimes compared to regular ALS is quite small in practice, even in absolute terms. The same algorithm applies to a much larger family of weighting schemes that can all be incorporated in time linear in the number of non-missing values.

Before going into detail, it is worth mentioning that the key property of ALS that enables this kind of optimization is its closed-form solution for computing optimal updates. Both pLSI and MMMF lack this property.

As our only constraint, we will assume that the weight matrix for the missing values can be expressed (or well approximated) by a low rank approximation: $\boldsymbol{W} = \boldsymbol{U}\boldsymbol{V}^T$. Positives automatically receive a weight of 1, regardless of the corresponding value in $\boldsymbol{U}\boldsymbol{V}^T$. All the weighting schemes proposed in [13] are covered as special cases of rank 1. Higher ranks allow for much more complex weighting schemes that can e.g., be motivated by domain knowledge or by iteratively adding latent features as part of a weight learning scheme. We leave the investigation of such schemes for future work.

To avoid confusion with the latent variables of $\boldsymbol{X}$ and $\boldsymbol{Y}$, referred to by using the summation variable d, we will use D as the variable for the latent features of $\boldsymbol{U}$ and $\boldsymbol{V}$, and in slight abuse of notation also denote the rank of $\boldsymbol{W}$ as $|D|$.

The novel gap-ALS algorithm (GALS) adapts Algorithm 1, but uses a much more efficient update rule than Eq. (2). We just show the case of updating $\boldsymbol{X}$ in Algorithm 2. Updating $\boldsymbol{Y}$ works analogously. Please refer to the appendix for a proof that Algorithm 2 optimizes $\boldsymbol{X}$ with respect to $\mathcal{L}_W$ when substituting AMAN and setting $W_{i,j} = 1$ for positives.

```
Require: matrices $\boldsymbol{R} \in \mathbb{R}^{n\times m}$, $\boldsymbol{Y} \in \mathbb{R}^{m\times d}$, $\boldsymbol{U} \in \mathbb{R}^{n\times|D|}$,
  $\boldsymbol{V} \in \mathbb{R}^{m\times|D|}$, substitute $W_{i,j} = 1$ if $R_{i,j} = 1$
Ensure: $\boldsymbol{X}$ with minimal loss $\mathcal{L}_W(\boldsymbol{X}, \boldsymbol{Y})$ for fixed $\boldsymbol{Y}$

  /* init */
  $\boldsymbol{V}_{\sum} := \boldsymbol{V}^T\mathbf{1}$
  for all $\tilde{D} \in \{1, \ldots, |D|\}$ do
    create new matrix $\widehat{\mathbf{A}}^{(\tilde{D})} \in \mathbb{R}^{d\times d}$ with:
    $\widehat{A}^{(\tilde{D})}_{j,k} := \sum_{i\in\mathcal{M}} Y_{i,j} \cdot Y_{i,k} \cdot V_{i,\tilde{D}}$

  /* row-wise recomputation of $\boldsymbol{X}$ */
  for rows $r \in \{1, \ldots, n\}$ do
    read non-missing value indices $\mathcal{M}_r$ for row $r$
    read rows of $Y$ in $\mathcal{M}_r$      (projection $\pi_r(\boldsymbol{Y})$)
    $\widehat{\mathbf{B}} := \pi_r(\boldsymbol{Y})^T\pi_r(\boldsymbol{Y})$
    $\mathbf{q_r} := \pi_r(\boldsymbol{Y})^T\mathbf{1}$
    read rows of $\boldsymbol{V}$ in $\mathcal{M}_r$      (projection $\pi_r(\boldsymbol{V})$)
    $\hat{c} := \boldsymbol{U}_r(\boldsymbol{V}_{\sum} - \pi_r(\boldsymbol{V})^T\mathbf{1})$
    for all $\tilde{D} \in \{1, \ldots, |D|\}$ do
      create new matrix $\widetilde{\mathbf{B}}^{(D)} \in \mathbb{R}^{d\times d}$ with:
      $\widetilde{B}^{(D)}_{j,k} := \sum_{i\in\mathcal{M}_r} Y_{i,j}Y_{i,k}V_{i,D}$
    read row-specific weight vector $\boldsymbol{U}_r$
    $\mathbf{A}' := \sum_D U_{r,D} \cdot \widehat{\mathbf{A}}^{(D)}$
    $\mathbf{B}' := \widehat{\mathbf{B}} - \sum_D U_{r,D} \cdot \widetilde{\mathbf{B}}^{(D)}$
    $\mathbf{C}' := \lambda(\hat{c} + |\mathcal{M}_r|) \cdot \mathbf{I}$
    $\boldsymbol{X}_r := \left[(\mathbf{A}' + \mathbf{B}' + \mathbf{C}')^{-1}\mathbf{q_r}\right]^T$
  return $\boldsymbol{X}$
```

Algorithm 2: GALS update of $\boldsymbol{X}$ given $\boldsymbol{R}$ and $\boldsymbol{Y}$ when weighting (only) missing values with $\boldsymbol{W} = \boldsymbol{U}\boldsymbol{V}^T$.

The runtime complexity of Algorithm 2 benefits substantially from precomputing the matrices $\widehat{A}^{(\tilde{D})}_{j,k}$ just once during the init-stage. The costs for this step, and for the whole initialization, are in $\Theta(|D| \cdot d^2 \cdot m)$. The costs for computing A', B', C' and q_r during each row-wise computation are low, because the computations are based on projections of matrices to $\mathcal{M}_r$. These costs are dominated by computing the $|D|$ matrices $\widetilde{B}^{(\cdot)}$, which takes time $\Theta(|D| \cdot d^2 \cdot |\mathcal{M}_r|)$ for each individual row r, or $\Theta(|D| \cdot d^2 \cdot |\boldsymbol{R}|)$ for all rows together. Finally we invert a $d \times d$ matrix for each row, as in regular ALS. The initialization phase is dominated by the row-wise updates. In the case of a rank 1 weighting, the asymptotic costs of GALS are identical to those of regular ALS, see Eq. (3). When using more complex weight matrices $\boldsymbol{W}$, GALS scales linearly with the rank of $\boldsymbol{W}$.

We want to conclude the discussion of runtime complexities by pointing out that GALS tends to converge quickly in practice due to the effective closed form updates. Even for large datasets, 20 to 30 iterations of updates are usually sufficient. In contrast, pLSI required hundreds of iterations in our experiments until convergence.

2.4 Maximum-Margin Matrix Factorization

In this section, we apply the idea of weighting unknowns into the context of Maximum Margin Matrix Factorization (MMMF) [17]. Given a matrix $\boldsymbol{R} = (R_{ij})_{m\times n} \in \{\pm 1\}^{m\times n}$ with m users and n items and a corresponding non-negative weight matrix $\boldsymbol{W} = (W_{i,j})_{m\times n} \in \mathbb{R}_+^{m\times n}$, weighted low-rank approximation aims at approximating $\boldsymbol{R}$ with a low rank matrix $\widetilde{\boldsymbol{R}} = (\widetilde{R}_{i,j})_{m\times n}$ minimizing the objective of the weighted hinge loss and the trace norm of $\widetilde{\boldsymbol{R}}$ as follows.

$$\mathcal{L}\left(\widetilde{\boldsymbol{R}}\right) = \sum_{i,j} W_{i,j} h\left(R_{i,j}, \widetilde{R}_{i,j}\right) + \lambda \left\|\widetilde{\boldsymbol{R}}\right\|_{\Sigma}, \tag{4}$$

where $h(\cdot)$ is the *Smooth Hinge* loss function in [17], λ is a parameter trading off the weighted hinge loss and the margin and $\|\cdot\|_\Sigma$ is the trace norm of a matrix. In the above objective function (Eq. (4)), $W_{i,j}$ reflects the contribution of minimizing the term to the overall objective $\mathcal{L}(\widetilde{\boldsymbol{R}})$.

Following the discussion in Subec. 2.3.2, we substitute negative labels (-1) for missing values and adopt the weighting schemes depicted in Table 1: Positive examples receive a fixed weight of $W_{i,j} = 1$, while the weights of negatives vary based on our confidence that they are in fact negative.

In order to solve the optimization problem $\min \mathcal{L}(\widetilde{\boldsymbol{R}})$, we consider the decomposition $\widetilde{\boldsymbol{R}} = \boldsymbol{X}\boldsymbol{Y}^T$ where $\boldsymbol{X} \in \mathbb{R}^{m\times d}$ and $\boldsymbol{Y} \in \mathbb{R}^{n\times d}$. Note that usually the number of features $d \ll r$ where $r \approx \min(m,n)$ is the rank of the matrix $\boldsymbol{R}$. We can re-write the objective function (Eq. (4)) as Eq. (5)

$$\mathcal{L}(\boldsymbol{X},\boldsymbol{Y}) = \tag{5}$$
$$\sum\nolimits_{i,j} W_{i,j}\left(h\left(R_{i,j}, \boldsymbol{X}_i \boldsymbol{Y}_j^T\right) + \frac{1}{2}\lambda\left(\|\boldsymbol{X}_i\|_F^2 + \|\boldsymbol{Y}_j\|_F^2\right)\right).$$

Taking partial derivatives of $\mathcal{L}$ with respect to each entry of $\boldsymbol{U}$ and $\boldsymbol{V}$, we obtain

$$\frac{\partial \mathcal{L}(\boldsymbol{X},\boldsymbol{Y})}{\partial X_{i,k}} = \sum\nolimits_j W_{i,j} h\left(R_{i,j}, \boldsymbol{X}_i \boldsymbol{Y}_j^T\right) Y_{j,k}$$
$$+\lambda\left(\sum\nolimits_j W_{i,j}\right) X_{i,k}, \forall 1 \le i \le m, 1 \le k \le d. \tag{6}$$

As suggested in [15], we apply the Conjugate Gradients (MMMF-CG) algorithm to solve the optimization problem $\min \mathcal{L}(\boldsymbol{X},\boldsymbol{Y})$. Note that the running time for wMMMF with CG is $\Theta(m \times n)$ however, if we substitute negatives for all missing values. Considering the large number of missing values, this is intractable for large-scale problems, and it may still cause issues related to class-imbalance, otherwise. Unlike for wALS, wMMMF updates do not have a closed-form solution. It is not clear how to speed up wMMMF in a similar manner as discussed in Subsec. 2.3.4. We hence adapt the previously discussed idea of subsampling (cf. Sec. 2.3.3).

Algorithm 3 summarizes this method. It combines the idea of subsampling unknowns as negatives with the bagging technique [5]. The algorithm runs iteratively, and each iteration i can be broken down into two phases.

In phase I, the algorithm samples a limited number of q negative examples from missing values, proportionately to our weight matrix $\boldsymbol{W}$. It then generates a new matrix $\boldsymbol{R}^{(i)}$ that includes the sample together with all positives from $\boldsymbol{R}$ (since positives are rare). For sparse matrices, the number of negative examples is close to $m \times n$, so care must be taken when selecting a negative example sampling (NES) technique to avoid complexities of $\Omega(m \times n)$. We suggest using a fast ($O(q)$) random sampling scheme [18] in these cases to generate $\boldsymbol{R}^{(i)}$ from $\boldsymbol{R}$.

In phase II, the algorithm re-constructs the rating matrix $\widetilde{\boldsymbol{R}}^{(i)}$ from $\boldsymbol{R}^{(i)}$ by applying a conjugate gradients algorithm (MMMF-CG) [15].

```
Require: matrix $\boldsymbol{R} \in \mathbb{R}^{m\times n}$, matrix $\widehat{\boldsymbol{W}} \in \mathbb{R}^{m\times n}$, sample
  size $q$, number of single predictor $\ell$
Ensure: Reconstructed matrix $\widetilde{\boldsymbol{R}}$
  for $i = 1 : \ell$ do
    Generate a new training matrix $\boldsymbol{R}^{(i)}$ by NES ([18])
    Reconstruct $\widetilde{\boldsymbol{R}}^{(i)}$ from $\boldsymbol{R}^{(i)}$ by MMMF-CG
  $\widetilde{\boldsymbol{R}} = \frac{1}{\ell}\sum_{i=1}^{\ell}\widetilde{\boldsymbol{R}}^{(i)}$
  return $\widetilde{\boldsymbol{R}}$
```

Algorithm 3: sMMMF-ENS Algorithm for OCCF

In the post-processing step, we compute an unweighted average of all the $\widetilde{\boldsymbol{R}}^{(i)}$ that were computed throughout the different iterations. This yields the final matrix $\widetilde{\boldsymbol{R}}$ which approximates $\boldsymbol{R}$. Bagging is promising in this setting, because it allows to exploit the stochastic, unstable nature induced by the underlying boostrap sampling approach. We refer to Algorithm 3 as *sampling MMMF Ensemble* (sMMMF-ENS).

3. EXPERIMENTS

In this section, we empirically evaluate the effectiveness and efficiency of the algorithms proposed in Section 2. We first study the predictive performance of all algorithms discussed in this paper on the same (small) datasets that were used in [13]. Then we study both predictive performance and algorithmic efficiency on three large-scale datasets. We use Mean Average Precision (MAP), Half-Life Utility (HLU) [13] and the Area Under the ROC curve (AUC) as our metrics for the predictiveness of models.

MAP (Mean Average Precision) assesses the overall performance based on precisions at different recall levels on a test set. It computes the mean of average precision (AP) over all users in the test set, where AP is the average of precisions computed at all positions with a preferred item:

$$AP_u = \frac{\sum_{i=1}^N prec(i) \times pref(i)}{\#\text{ of preferred items}}, \tag{7}$$

where i is the position in the rank list, N is the number of retrieved items, $prec(i)$ is the precision (fractions of retrieved items that are preferred by the user) of a cut-off rank list from 1 to i, and $pref(i)$ is a binary indicator returning 1 if the i-th item is preferred or 0, otherwise.

Half-Life Utility (HLU) [4] estimates how likely a user will view/choose an item from a ranked list, which assumes that the user will view each consecutive item in the list with an exponential decay of possibility. A half-life utility over all users in a test set is defined as in Eq. (8).

$$R = 100\frac{\sum_u R_u}{\sum_u R_u^{max}}, \tag{8}$$

where R_u is the expected utility of the ranked list for user u and R_u^{max} is the maximally achievable utility if all true positive items are at the top of the ranked list. R_u is defined as follows.

$$R_u = \sum_j \frac{\delta(j)}{2^{(j-1)(\beta-1)}}, \tag{9}$$

where $\delta(j)$ equals 1 if the item at position j is preferred by the user and 0 otherwise, and β is the half-life parameter which is set to 5 in this paper.

Table 2: Comparisons of MAP, HLU and AUC on Large-Scale Datasets

	newsSmall			newsLarge			Netflix		
	MAP	HLU	AUC	MAP	HLU	AUC	MAP	HLU	AUC
POP	0.62	1.09	0.736	0.50	0.90	0.842	8.40	15.52	0.936
SVD	2.57	5.13	0.88	1.97	4.16	0.851	13.28	25.69	0.86
GALS-Uniform	3.28	5.89	0.90	2.37	4.53	0.854	16.82	**28.94**	0.957
GALS-User	3.28	6.04	0.907	2.38	4.47	0.852	**16.87**	28.81	0.958
GALS-Item	**3.30**	**6.05**	0.908	**2.39**	**4.54**	0.849	16.79	28.85	0.958
sMMMF-ENS	2.67	5.24	**0.913**	1.39	2.85	**0.884**	11.73	19.04	**0.961**
pLSI	0.65	0.55	0.89	0.11	0.06	0.78	4.42	7.33	0.862

3.1 Predictive Performances on Small Data

The following experiments compare the predictive performances of GALS, wMMMF, and some baseline algorithms. For the purpose of comparisons, we use the same datasets as in [13]. This includes data from the news domain (Yahoo! News usage) and from the domain of social bookmarking (delicious.com). For validation purposes, each dataset contains 20 pairs of training and test sets, each with an 80/20 splitting ratio. The Yahoo! data contains 84117 user-article pairs with 3158 unique users and 1536 different articles. The delicious.com data contains 246, 436 posts by 3000 users, and it covers 2000 different tags.

Figure 1 shows results in terms of MAP, HLU and AUC, averaged over the 20 folds. SVD (with all missing as negative substitution) and Popularity are two baselines also used in [13]. As expected, wMMMF performs best with respect to both MAP and HLU and is competitive for AUC, which verifies that the maximum-margin paradigm can also achieve outstanding performance on classification problems in collaborative filtering setting. GALS yields competitive results with all three weighting schemes. Surprisingly, pLSI works poorly for OCCF. One possible reason is that it implicitly makes the "all missing as unknown" assumption.

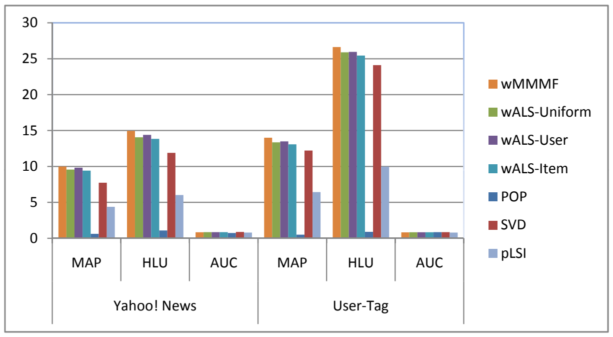

Figure 1: GALS vs. mMMMF

3.2 Large-Scale OCCF Experiments

3.2.1 Experimental setup

We used three large test datasets to compare our proposed algorithms to state-of-the-art baseline algorithms. The first one is a clickstream dataset of News articles[1]. Each record is a user-article pair that consists of a user ID and the URL of a NYTimes, Washington Post, or Yahoo news article. The data set contains 59202 unique users and 138, 575 different news stories. We call this dataset *newsLarge* in our experiments. We also generated a smaller dataset from newsLarge, such that each user or each item has at least ten records. We call this dataset *newsSmall*. It contains 23, 495 users and 36, 339 articles. The third dataset is generated from the Netflix Prize data. It contains about 100 million ratings with 480, 189 users and 17, 770 movies. The ratings in the original data ranges from 1 to 5, but we just tried to predict whether a user rated a movie at all. That is, we treat all rated user-movie pairs as positive examples, and all pairs without given ratings as missing values.

We randomly divided each of the three datasets into a training, a validation and a test set with a 60/20/20 splitting ratio. The training set contains 60% known positive examples. The other elements of the matrix are treated as unknown. The validation set includes 20% known positives, and all the unknown examples for parameter tuning. The test set includes the remaining 20% known positives and all unknown examples.

Note that the known positives in the training set are excluded from the test process. We evaluate the performance on the test set using AUC, MAP, and half-life utility (HLU). The parameters of all algorithms were determined by the performance on validation set.

3.2.2 Predictive Performances

We conducted experiments on our three large datasets to demonstrate the utility of the proposed algorithms for OCCF. The algorithms involved in these experiments include Popularity, SVD (AMAU), pLSI, sMMMF-ENS (Algorithm 3)[2], and GALS with the three weighting schemes discussed in Section 2.3.2. We fixed the number of features to 50 throughout these experiments. Table 2 lists the results. Although sMMMF-ENS achieves highest AUC values, it is not as good as wMMMF (shown in Figure 1) on MAP and HLU. In contrast, GALS with different weighting schemes outperforms other algorithms on all three datasets, which shows great merit in solving large-scale OCCF problems. We will also illustrate the advantages of GALS in terms of time complexity.

3.2.3 Tests of Computational Costs

We implemented the GALS algorithm in Matlab and Java to confirm that it gives identical results as the less scalable counter-part and to study how it scales up as a function of different variables. The results we report here were gen-

[1]We thank "NielsenOnline" for providing the clickstream data.

[2]wMMMF is not included because it is intractable for large-scale datasets, although it performs well in the above experiments.

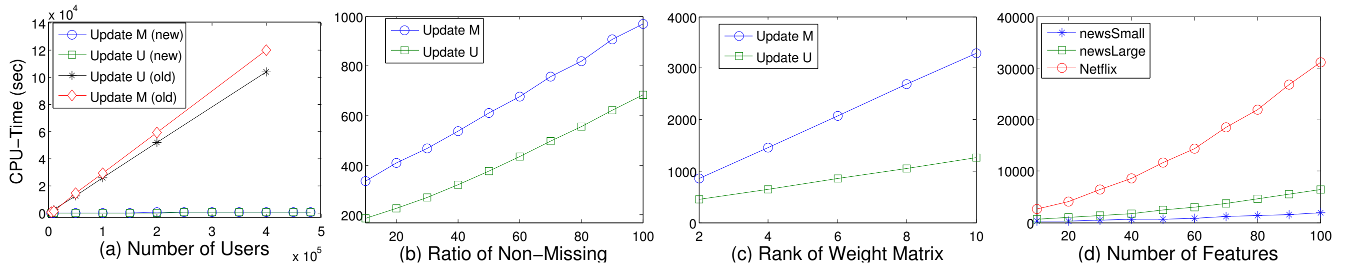

Figure 3: (a) Changing CPU time for a fixed number of items and a varying number of users for the explicit full substitution (old) and our new method; (b) Changing CPU time for a fixed matrix if we only vary the sparsity; (c) Changing CPU time as a function of an increasing rank of the weight matrix; (d) Changing CPU time as a function of an increasing number of features.

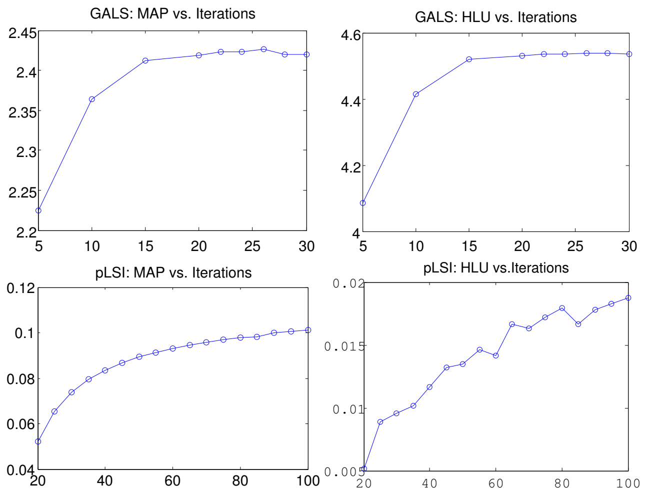

Figure 2: Convergence: GALS vs. pLSI

erated using the Matlab code on the binary (who ranked what?) Netflix data set. The sparsity of the data is about 1%, there are about $20,000$ movies and $500,000$ users.

Figure 2 shows the convergence of GALS and pLSI. We can see that GALS converges after roughly 15 to 20 iterations, much earlier than pLSI.

Figure 3 (a) compares the time consumed by the updates for the old method that performs a full substitution of missing values and weights the whole matrix, and the new GALS update method. "Updating U" refers to an update of the user matrix (corresponds to X), while "updating M" corresponds to the item matrix (Y). We only evaluated the case of rank 1 weight matrices here. We failed to run the old method for more than $400,000$ users. It is obvious that the new method brings incredible savings in terms of CPU costs.

In Figure 3 (b), we fixed the matrix but removed positives at random. This allows us to increase the sparsity from 1% positives (right) to 0.1% left. Since we used the full matrix, we could only run our new method on this set. The plot confirms the linearity of our method in the number of non-missing values.

Figure 3 (c) confirms that increasing the rank of the weight matrix affects the computational costs only linearly.

Finally, Figure 3 (d) confirms that the number of features has an impact of square order on CPU time as analyzed in Section 2.3.

4. RELATED WORK

Many researchers have explored different aspects of collaborative filtering (CF) in the past, ranging from improving the performance of algorithms to incorporating more resources from heterogeneous data sources [1]. However, collaborative filtering research still assumes that there are explicit positive (high rating) and negative (low rating) examples and the goal is to distinguish between those two, which is different from the one-class case studied in this paper.

Missing values (e.g., ratings) are a common issue in CF. In [2] and [12], the authors discuss the issue of modeling the distribution of missing values. Both approaches cannot handle cases in which no negative examples are given.

In the binary case, each example is either positive or negative. Das et al. [7] describe a news recommendation problem in which clicks on news stories are interpreted as positive examples, and "non-clicks" as negative examples. The authors compare some practical methods on this large scale binary CF problem, which is close to the one-class case. The KDD Cup 2007 task was a "Who rated What" prediction problem based on the Netflix Prize dataset. We included a similar prediction problem in our experimental study. The winning team [10] proposed a hybrid method that combined SVD and popularity using binary training data.

Our work also has some commonalities with the class imbalance literature. Class-imbalance is typically studied for classification tasks. The two strategies that are commonly used to solve the problem are sampling to re-balance the data [3, 11] and cost-sensitive learning [8, 19]. A comparison of the two strategies can be found in [14].

5. CONCLUSIONS AND FUTURE WORK

We studied the question which matrix decomposition techniques work well on large-scale one-class collaborative filtering techniques. This question can be decomposed into two major parts: Which techniques have a good predictive performance, and how well do the techniques scale up. Our baseline was a popular probabilistic method, which we compared to low-rank approximation and maximum-margin methods. We recently demonstrated that weighting schemes are very valuable in this area, so we were also concerned with the question of how to incorporate the ability for weighting the unknown parts of a matrix.

The results without the scalability aspect were close to our expectation: The maximum-margin method performed very well, closely followed by ALS. The only surprise was the poor performance of pLSI. Weighting the gaps clearly helped to further improve the results for both MMMF and ALS. To the best of our knowledge, we ran the first experiments with a weighted MMMF variant. As another theoretical contribution, we proved that there is an exact, yet scalable updating mechanism for ALS for large-scale data sets that allows to weight all the gaps, as long as the corresponding weight matrix has a good or exact low-rank approximation. In contrast, at a large scale, MMMF can only make use of weights via a subsampling / ensemble technique, because the hinge loss is too complex to allow for adaptations similar to those we incorporated into ALS. We empirically showed that the subsampling work-around does not work as well as using the full set of weighted negatives, so MMMF lost the edge and the novel GALS algorithm excelled as the method of choice in our large-scale experiments.

In our future research, we plan to experiment with more complex weighting schemes and to learn such schemes iteratively.

6. REFERENCES

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

[2] Y. Azar, A. Fiat, A. R. Karlin, F. McSherry, and J. Saia. Spectral analysis of data. In *STOC*, pages 619–626. ACM, 2001.

[3] G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard. A study of the behavior of several methods for balancing machine learning training data. *SIGKDD Explorations*, 6(1):20–29, 2004.

[4] J. S. Breese, D. Heckerman, and C. M. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *UAI*, pages 43–52. Morgan Kaufmann, 1998.

[5] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[6] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9:251–280, 1990.

[7] A. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. In *WWW*, pages 271–280. ACM, 2007.

[8] C. Drummond and R. Holte. C4.5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling. In *Proc. ICML'2003 Workshop on Learning from Imbalanced Data Sets II*, 2003.

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

[10] M. Kurucz, A. A. Benczur, T. Kiss, I. Nagy, A. Szabo, and B. Torma. Who rated what: a combination of SVD, correlation and frequent sequence mining. In *Proc. KDD Cup and Workshop*, 2007.

[11] X.-Y. Liu, J. Wu, and Z.-H. Zhou. Exploratory under-sampling for class-imbalance learning. In *ICDM*, pages 965–969. IEEE Computer Society, 2006.

[12] B. Marlin, R. Zemel, S. Roweis, and M. Slaney. Collaborative filtering and the missing at random assumption. In *UAI*, 2007.

[13] R. Pan, Y. Zhou, B. Cao, N. N. Liu, R. Lukose, M. Scholz, and Q. Yang. One-class collaborative filtering. In *IEEE International Conference on Data Mining (ICDM)*, 2008.

[14] B. Raskutti and A. Kowalczyk. Extreme re-balancing for SVMs: a case study. *SIGKDD Explorations*, 6:60–69, 2004.

[15] J. D. M. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In L. D. Raedt and S. Wrobel, editors, *ICML*, volume 119 of *ACM International Conference Proceeding Series*, pages 713–719. ACM, 2005.

[16] N. Srebro and T. Jaakkola. Weighted low-rank approximations. In *International Conference on Machine Learning (ICML)*, 2003.

[17] N. Srebro, J. D. M. Rennie, and T. Jaakkola. Maximum-margin matrix factorization. In *NIPS*, 2004.

[18] J. S. Vitter. Faster methods for random sampling. *Commun. ACM*, 27(7):703–718, 1984.

[19] G. M. Weiss. Mining with rarity: a unifying framework. *SIGKDD Explorations*, 6:7–19, 2004.

[20] C. V. Yehuda Koren, Yifan Hu. Collaborative filtering for implicit feedback datasets. In *IEEE International Conference on Data Mining (ICDM)*, 2008.

[21] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan. Large-scale parallel collaborative filtering for the Netflix Prize. In *Proc. of Algorithmic Aspect of Information and Management (AAIM'08)*, 2008.

APPENDIX

Correctness of GALS (Algorithm 2).

We start from the objective function and break it down into parts. For notational simplicity and brevity we omit some arguments if clear from the context, and define $\mathcal{L}^*$, A through C, $\mathbf{A}'$ through $\mathbf{C}'$, and $\mathbf{q_r}$ inline.

$$\begin{aligned}
\mathcal{L}_W(\boldsymbol{X},\boldsymbol{Y}) &= \sum_r \mathcal{L}_W(\boldsymbol{X}_r,\boldsymbol{Y}) \\
\mathcal{L}_W(\boldsymbol{X}_r,\boldsymbol{Y}) &= \underbrace{\sum_{i\in\mathcal{M}} W_{r,i}(X_rY_i^T - R_{r,i})^2}_{=:\mathcal{L}^*(\boldsymbol{X}_r,\boldsymbol{Y})} \\
&+ \underbrace{\sum_{i\in\mathcal{M}} W_{r,i}\lambda(||\boldsymbol{X}_r||_F^2 + ||\boldsymbol{Y}_i||_F^2)}_{=:C}
\end{aligned}$$

We now simplify the row-wise unregularized loss $\mathcal{L}^*(\boldsymbol{X}_r,\boldsymbol{Y})$, exploiting that $R_{r,i}=1$ and $W_{r,i}=1$ whenever $i\in\mathcal{M}_r$ (non-missing) and $R_{r,i}=0$, otherwise. Then we decompose it into a (mostly) global part A and a row-dependent B:

$$\begin{aligned}
&\mathcal{L}^*(\boldsymbol{X}_r,\boldsymbol{Y}) \\
=& \sum_{i\in\mathcal{M}} W_{r,i}\cdot(\boldsymbol{X}_r\boldsymbol{Y}_i^T - R_{r,i})^2 \\
=& \sum_{i\in\mathcal{M}\setminus\mathcal{M}_r} \boldsymbol{U}_r\boldsymbol{V}_i^T(\boldsymbol{X}_r\boldsymbol{Y}_i^T)^2 + \sum_{i\in\mathcal{M}_r}(\boldsymbol{X}_r\boldsymbol{Y}_i^T - 1)^2
\end{aligned}$$

$$= \underbrace{\sum_{i\in\mathcal{M}} \boldsymbol{U}_r\boldsymbol{V}_i^T(\boldsymbol{X}_r\boldsymbol{Y}_i^T)^2}_{=:A}$$

$$+ \underbrace{\sum_{i\in\mathcal{M}_r}\Big((\boldsymbol{X}_r\boldsymbol{Y}_i^T-1)^2-\boldsymbol{U}_rV_i^T(\boldsymbol{X}_r\boldsymbol{Y}_i^T)^2\Big)}_{=:B}$$

After this decomposition, we are interested in the derivatives

$$\partial\mathcal{L}(\boldsymbol{X}_r,\boldsymbol{Y})/\partial X_{r,c}=\frac{\partial A}{\partial X_{r,c}}+\frac{\partial B}{\partial X_{r,c}}+\frac{\partial C}{\partial X_{r,c}}$$

for all components $X_{r,c}$ of vector $\boldsymbol{X}_r$ to set them to 0. This we do in the form of a linear equation system, where each component (value of $c\in\{1,\dots,d\}$) corresponds to another line. The equation system will be of the form $(\mathbf{A'}+\mathbf{B'_r}+\mathbf{C'})\boldsymbol{X}_r^T=\mathbf{q_r}$, where $\mathbf{A'}\boldsymbol{X}_r^T$ through $\mathbf{C'}\boldsymbol{X}_r^T$ correspond to the partial derivatives of A through C. We start with $\mathbf{A'}$.

$$\begin{aligned}\frac{\partial A}{\partial X_{r,c}} &= 2\sum_{i\in\mathcal{M}}\Big(Y_{i,c}\sum_D(U_{r,D}\cdot V_{i,D})(\sum_d X_{r,d}\cdot Y_{i,d})\Big)\\ &= 2\sum_{i\in\mathcal{M}}\Big(Y_{i,c}\sum_D\sum_d U_{r,D}\cdot V_{i,D}\cdot X_{r,d}\cdot Y_{i,d}\Big)\\ &= 2\sum_d\sum_D\sum_{i\in\mathcal{M}}(Y_{i,c}\cdot U_{r,D}\cdot V_{i,D}\cdot X_{r,d}\cdot Y_{i,d})\\ &= 2\sum_d X_{r,d}\sum_D U_{r,D}\sum_{i\in\mathcal{M}}(Y_{i,c}\cdot Y_{i,d}\cdot V_{i,D})\end{aligned}$$

Based on the inner sum, we define $|D|$ (the rank of the weight matrix W) two-dimensional matrices $\widehat{\mathbf{A}}^{(1)},\dots,\widehat{\mathbf{A}}^{(|D|)}$:

$$\widehat{A}_{c,d}^{(\tilde{D})}:=\sum_{i\in\mathcal{M}}Y_{i,c}\cdot Y_{i,d}\cdot V_{i,\tilde{D}}$$

We can now define $\mathbf{A'}$ by weighting the $|D|$ matrices according to the row-specific weight vector $\boldsymbol{U}_r$:

$$A':=\sum_D U_{r,D}\cdot\widehat{\mathbf{A}}^{(D)}.$$

We already dropped the factor of 2 that is shared by $\mathbf{B'_r}$, $\mathbf{C'}$, and $\mathbf{q_r}$ and cancels out later. The partial derivative of B is:

$$\begin{aligned}\frac{\partial B}{\partial X_{r,c}} &= \left[\partial\sum_{i\in\mathcal{M}_r}\Big((\boldsymbol{X}_r\boldsymbol{Y}_i^T-1)^2-\boldsymbol{U}_r\boldsymbol{V}_i^T(\boldsymbol{X}_r\boldsymbol{Y}_i^T)^2\Big)\right]/\partial X_{r,c}\\ &= \sum_{i\in\mathcal{M}_r}\Big(2Y_{i,c}\boldsymbol{X}_r\boldsymbol{Y}_i^T-2Y_{i,c}-2Y_{i,c}\boldsymbol{U}_r\boldsymbol{V}_i^T\boldsymbol{X}_r\boldsymbol{Y}_i^T\Big)\end{aligned}$$

After removing factor of 2,

$$\sum_d X_{r,d}\sum_{i\in\mathcal{M}_r}\Big(Y_{i,c}Y_{i,d}-Y_{i,c}Y_{i,d}\sum_D U_{r,D}V_{i,D}\Big)-\sum_{i\in\mathcal{M}_r}Y_{i,c}$$

$$=\sum_d X_{r,d}\left(\underbrace{(\sum_{i\in\mathcal{M}_r}Y_{i,c}Y_{i,d})}_{\widehat{=}\widehat{B}}-(\sum_D U_{r,D}\underbrace{\sum_{i\in\mathcal{M}_r}Y_{i,c}Y_{i,d}V_{i,D})}_{\widehat{=}\widetilde{B}_{c,d}^{(D)}}\right)$$

$$\underbrace{-\sum_{i\in\mathcal{M}_r}Y_{i,c}}_{\widehat{=}q_r}$$

The matrix form to be plugged into the equation system is:

$$\begin{aligned}\mathbf{B'} &:= \widehat{\boldsymbol{B}}-\sum_D U_{r,D}\cdot\widetilde{\mathbf{B}}^{(D)}\text{ , where}\\ \widehat{B} &:= \pi_r(Y)^T\pi_r(Y)\\ \widetilde{B}_{c,d}^{(D)} &:= \sum_{i\in\mathcal{M}_r}Y_{i,c}Y_{i,d}V_{i,D}\\ \mathbf{q_r} &:= \pi_r(\boldsymbol{Y})^T\mathbf{1}\end{aligned}$$

The regularization part C can be rewritten as

$$C = \Big(\sum_d X_{r,d}^2+Y_{r,d}^2\Big)\lambda\Big(\sum_{i\in\mathcal{M}\setminus\mathcal{M}_r}\boldsymbol{U}_r\boldsymbol{V}_i^T+\sum_{i\in\mathcal{M}_r}1\Big)$$

A trivial multiplication of $\boldsymbol{U}$ and $\boldsymbol{V}$ leads to non-linear costs, so we instead decompose the derivative

$$\begin{aligned}&\frac{1}{2}\frac{\partial C}{\partial X_{r,c}}\\ &= X_{r,c}\lambda\Big((\sum_{i\in\mathcal{M}}\boldsymbol{U}_r\boldsymbol{V}_i^T)-(\sum_{i\in\mathcal{M}_r}\boldsymbol{U}_r\boldsymbol{V}_i^T)+|\mathcal{M}_r|\Big)\\ &= X_{r,c}\lambda\left[|\mathcal{M}_r|+\sum_D U_{r,D}\Big((\sum_{i\in\mathcal{M}}V_{i,D})-(\sum_{i\in\mathcal{M}_r}V_{i,D})\Big)\right]\end{aligned}$$

and compute the corresponding matrix $\mathbf{C'}$ as

$$\begin{aligned}\mathbf{C'} &:= \lambda(\hat{c}+|\mathcal{M}_r|)\cdot\mathbf{I}\text{, where}\\ \hat{c} &:= \boldsymbol{U}_r(\boldsymbol{V}_\Sigma-\pi_r(\boldsymbol{V})^T\pi_r(\mathbf{1}))\\ \boldsymbol{V}_\Sigma &:= \boldsymbol{V}^T\mathbf{1}\end{aligned}$$

Now we can plug in in the partial results used by Algorithm 2 and confirm that $(\mathbf{A'}+\mathbf{B'}+\mathbf{C'})^{-1}\mathbf{q_r}$ in fact solves the linear equation system we get when setting all partial derivatives to 0:

$$\begin{aligned}&\boldsymbol{X}_r^T=(\mathbf{A'}+\mathbf{B'}+\mathbf{C'})^{-1}\mathbf{q_r}\\ \Leftrightarrow\ &(\mathbf{A'}+\mathbf{B'}+\mathbf{C'})\boldsymbol{X}_r^T=\mathbf{q_r}\\ \Leftrightarrow\ &\forall c:(\frac{\partial A}{\partial X_{r,c}}+\frac{\partial B}{\partial X_{r,c}}+\frac{\partial C}{\partial X_{r,c}})=0\\ \Leftrightarrow\ &\forall c:\partial\left[\mathcal{L}^*(\boldsymbol{X}_r,\boldsymbol{Y})+C\right]/\partial X_{r,c}=0\\ \Leftrightarrow\ &\forall c:\partial\left[\mathcal{L}_W(\boldsymbol{X}_r,\boldsymbol{Y})\right]/\partial X_{r,c}=0\end{aligned}$$

Due to the regularization term, the matrix inversion is always well-defined. It can easily be seen that the overall optimization problem is still convex, so Algorithm 2 computes the global optimum for each $\boldsymbol{X}_r$, and hence for $\boldsymbol{X}$. **q.e.d.**