Master Thesis

Econometrics and Management Science

Business Analytics and Quantitative Marketing

Sjoerd Janssen

Spectral Regularization Algorithms for Recommender Systems with Large Implicit Datasets

Submitted to:
Prof. dr. P.J.F. Groenen
Full Professor
Department of Econometrics

Submitted by:
Sjoerd Janssen
447501
.

Contents

Preface and Acknowledgements

When I started my thesis-internship 7 months ago at Helloprint, my main focus was to find an interesting topic that was academically relevant as well as of practical use to the company where I did my thesis.

Purely from a personal point of view, I was already interested in the mechanisms behind recommender systems. When I listen to Spotify, I am enthusiastic about the new songs that are suggested to me. On the other hand, while watching movies, I am annoyed about the small amount of variation in Netflix's recommendations.

When I noticed that little research was done on recommender systems for implicit data, I quickly realized that I found the perfect topic. This was academically relevant, useful for Helloprint and it was something that helped me to understand the mechanisms that provided me with both irritations and moments of joy.

I did not yet realize that I also found a topic that challenged me to push my boundaries, nor that it was the beginning of a project that felt like a roller-coaster ride. There were moments where it felt like I had accomplished the impossible by completing a proof, followed by disappointments when I realized that I made a small error in my calculations. However, in the end, when everything has worked out, you only remember the fun parts and you can look back on a hell of a ride. I am grateful to the people that helped me enjoy it.

In particular, I want to thank prof. dr. Patrick Groenen for the supervision during this project. During our meetings, he provided me with new directions we could take, he suggested papers that I could use to as inspiration in my work and he provided me with useful feedback on my work in progress. Almost every meeting was equally inspiring and pushed me to take my work to the next level.

I also want to thank all my colleagues at Helloprint for all the fun we had and for the pleasant collaboration. A special thanks to the Data Science and BI team, with whom we have improved so many processes for the company. A je to!

Next, I want to thank my family and friends. Thanks for listening to the stories about the work I was occupied with. I know you are not as enthusiastic about matrix factorization as I am, but your listening ears and motivating words really helped me to achieve my goals for this thesis. Also, thank you for distracting me from my work. Thank you for taking me on holiday, starting a foodtruck with me, dragging me to parties and taking me out to dinner. Without those distractions, I would not have had the focus that I needed during the workweek.

And last, but not least, I want to thank you. Thank you for reading the first page of this paper. If you are planning to continue reading, I hope you enjoy it as much as I did.

Kind regards,
Sjoerd Janssen 24-10-2017, Amsterdam

Abstract

Recommender systems are a class of algorithms that help customers to overcome their choice problem by recommending products that they are likely to be interested in. Most recommender systems are developed for datasets where users explicitly provide feedback in the form of ratings. In practice, such datasets are often unavailable and companies have to resort to implicit datasets, where feedback is distilled from different sorts of user's behavior, such as click through data or purchases. There exists some models that are specifically suited for implicit datasets and that can be estimated with alternating least squares (ALS) and stochastic gradient descent methods. The main goal of this research is to develop convex spectral regularization algorithms that are able to solve these implicit models. Until now, such algorithms are only designed for explicit context and cannot be applied to solve implicit models. The advantage of these algorithms is that they are computationally very efficient and need fewer observations than their ALS counterparts. We develop three algorithms that optimize three different implicit data models. One of these models is newly developed and based on a squared hinge loss function. All algorithms scale linearly in complexity with the size of the data. From a simulation study, we find evidence that the algorithms perform well for large datasets. We successfully apply the algorithms to an empirical dataset of an e-commerce company and show that they compare favourably to competing methods. We find no evidence for superior performance of our newly developed one-sided squared hinge loss model.

Keywords: Spectral regularization, Implicit data, Matrix Factorization, Nuclear Norm

1 Introduction

The growing popularity of e-commerce can be explained by the ease of ordering and their broad range of products. One of the major challenges, however, is to help customers select the relevant products out of all these offers. Where offline stores have the advantage of personnel that helps the customers, e-commerce companies have to resort to other solutions. One approach to help customers with this choice overload problem is to provide them with personalized recommendations. Since recently, many variations of computer algorithms are developed to deal with this problem. These so called "recommender systems" select and display a group of products that, hopefully, fit the customers' wants and needs.

All recommender systems aim to relate products and users, but their strategies differ. Roughly, there are two main strategies: *content-based filtering* and *collaborative filtering*. Combinations of these two are also possible. In content-based filtering,

attributes are assigned to either the user (customer) or the item (product) in order to profile them. The profiles that are formed by these attributes are used to associate users with matching products. For collaborative filtering algorithms, the main focus of this research, the recommendations are based on past feedback from customers and the creation of such an explicit profile is unnecessary. These systems generally outperform pure content-based techniques (Koren et al., 2009).

Collaborative filtering algorithms can again be subdivided into *neighborhood methods* and *latent factor models*. The first type aims to find users (or items) that show strong similarity in terms of their ratings (the neighbors). It will then recommend items based on the ratings of these neighbors. Latent factor models assume that ratings are deeply influenced by a (relatively) small set of domain specific latent factors. These latent factors together span a latent feature space. The method seeks to find vectors for both the user and the item in this space. The entries of these vectors will represent the score of the user or item on that specific factor and their dot product will be an approximation of the rating. Interpretation of these factors is generally not obvious since they are mathematically derived. For example, a latent factor for movie recommendation could represent something like its genre, but also a measure for the positive vibe in a movie or a combination of both.

Matrix Factorization (MF) techniques are a class of widely successful latent factor models that aim to find low-rank approximations to the user-item matrix, a matrix with all known interactions between users and items. These low rank approximations will then consist of the user and item vectors.

MF models became widely popular since the Netflix Prize competition (Bennett et al., 2007) and many authors have suggested algorithms that solve these models. Even though there exists an abundance of research on this topic, almost all MF recommender systems use high quality *explicit feedback* as input. Explicit feedback consists of users' ratings for items, for example with a 5 stars rating mechanism or a thumb up / thumb down system. Explicit feedback is convenient as it incorporates a lot of information. However, in practice, it is often unavailable and business have to resort to *implicit feedback*. This type of feedback is distilled from user behavior and indirectly reflects a users preferences. Examples of implicit feedback include purchase history, mouse movements, search patterns and clicks on a website. Because of the practical relevance, this research will focus on MF techniques for implicit feedback.

Implicit feedback is fundamentally different from explicit feedback. Hu et al. (2008) identify four main differences. First, there is no clear negative feedback; having not clicked a product does not necessarily imply that a user does not like the item, he could simply be unaware that this product is offered. Second, the data is inherently noisy; having clicked a product does not necessarily imply positive feedback. It could also be an accidental click. Third, the numerical value does not

indicate the preference of a user. For instance, more purchases don't necessarily imply the user is more positive about the product. It could be that the user simply needs repeat purchases for that product. The numerical value is, however, correlated with the confidence we have in positive feedback. The final difference they note is that implicit feedback calls for a different evaluation metric. We cannot compare the predicted rating with the numerical value of implicit data and should therefore resort to another, more appropriate, metric.

Some authors developed methods for recommender systems that are specifically suited to the implicit data context (Hu et al., 2008; Johnson, 2014; Koren, 2008; Lee et al., 2008; Pilászy et al., 2010). Their approaches are based on a variety of different ideas such as the inclusion of temporal information or incorporating a confidence measure in an effort to improve accuracy. For optimization, they often use Gradient Descent or Alternating Least Squares methods. In Mazumder et al. (2010), an attractive alternative is introduced to these optimization methods. The authors develop the `softImpute` algorithm, which is very efficient in handling large matrices. This convex spectral regularization algorithm introduced in Mazumder et al. (2010) seems to be faster than the available Gradient Descent Methods and from Jain et al. (2013), we know that convex minimization methods need fewer observed data points than alternating least squares methods to arrive at the correct solution.

Because the `softImpute` algorithm only works with explicit data, we aim to extend it to the implicit data context. Hence, this research answers the following main research question: *Can we develop a convex spectral regularization algorithms for recommender systems with large implicit datasets?* In this research, we develop three of such algorithms. For each of these algorithms, we also investigate how well they perform in terms of speed, their ability to uncover the true underlying preferences and how they compare to other methods. This paper does not aim to evaluate the performance of existing methods since this is often very dependent on the situation. Instead, we aim to provide new flexible efficient algorithms that can be used in conjunction with existing methods.

In order to answer our research question, we organize this paper as follows. First, we discuss the methods of dealing with implicit data. We review some literature and examine the existing models that are designed for implicit feedback data. In addition, we develop a new one-sided squared hinge loss model. Second, the optimization is discussed. We examine the algorithm of Mazumder et al. (2010) and develop three algorithms for the implicit data case. In the third section, we design and conduct a simulation study to investigate the performance of our algorithms. In the fourth section, all algorithms are tested on an empirical dataset and we will illustrate that the new algorithms show superior performance to current methods.

2 Dealing with Implicit Data

2.1 Related Works

There is little research on dealing with implicit data in recommender systems. Since Oard et al. (1998) laid out the groundwork on different strategies that researchers could use to transform implicit feedback into predicted ratings, only few authors have actually designed such systems.

Perhaps the most cited work is from Hu et al. (2008), who distinguishes two parts that are embedded in a user's implicit feedback data. Namely, the positive- or no-feedback for the product and the confidence we have in this feedback. They define positive- or no-feedback as a binary indicator and they suggest several ways to transform the frequency of the implicit data into a confidence measure. They then suggests to use these confidence measures as weights on the squared errors, assigning a higher weight to the losses where we are more certain. Their proposed algorithm solves the weighted problem by means of Alternating Least Squares (ALS).

Pan et al. (2008) has simultaneously designed a weighted ALS algorithm to deal with implicit feedback. However, they argue that no-feedback should be treated as negative feedback with varying but lower weights. They suggest, for example, that if a user has already viewed many products, we can infer that he does not like the products without feedback with a higher probability. Therefore we assign higher weights to their negative feedback. Simultaneously, they treat all implicit feedback as binary positive feedback with the same weights.

Other authors have suggested to incorporate even more information through different weighting schemes. For example, Lee et al. (2008) use time dependent weights in their nearest neighbor model in order to account for preferences that change over time. Also, Johnson (2014) suggests that for a music streaming service, streams with explicit clicks should be weighted higher than streams without clicks. Finally, Fang and Si (2011) add weighting schemes that include user-item similarity for situations in which there is rich information on both users as well as items.

Johnson (2014) extends the method of Hu et al. (2008) by exploiting the binary structure of the positive feedback. He presents the Logistic Matrix Factorization algorithm, where he follows a probabilistic approach to explain each matrix entry. He uses the same weighting scheme as Hu et al. (2008) but then applied on the likelihood contributions of each observation in order to estimate the model. The model is estimated by a alternating gradient descent approach. His model outperforms Hu et al. (2008) in terms of accuracy.

In summary, in dealing with implicit data, most authors aim to incorporate some form of information in the model through weights in the cost function. Then they proceed to estimate their model through either alternating least squares or gradient descent methods.

2.2 Notation and Problem Formulation

We build upon the strategy of Hu et al. (2008) of distinguishing a feedback and confidence part in the data. By doing so, we can distinguish two cases:

Case 1: Feedback is provided. We assume that the user has a positive preference for the item. These observations have, by definition, a larger weight than observations for which no feedback is provided. The weights express the confidence we have in this assumption and are correlated with the numerical value of the implicit feedback.

Case 2: Feedback is not provided. We typically do not know why a user has not provided feedback for the item. Reasons include, but are not limited to, dislike, unawareness and no need for a product. We assume that the user has neutral preference. We assign a relatively low weight to these observations. The low weight reflects the low confidence we have in this assumption.

If we collect all the information on all interactions between users $i = 1, \ldots, n$ and items $j = 1, \ldots, p$, we can construct $n \times p$ matrix $\boldsymbol{X}$ whose entries x_{ij} denote the feedback of user i on product j. For Implicit feedback, we set $x_{ij} = 1$ and $x_{ij} = 0$ for Case 1 and 2 respectively. It is typical for matrix $\boldsymbol{X}$ to be very sparse, since most users only provide feedback for a few products.

It is our objective to explain matrix $\boldsymbol{X}$ with some $\boldsymbol{AB'}$ where $\boldsymbol{A}$ is a $n \times k$ and $\boldsymbol{B}$ is a $p \times k$ matrix, with $k < \min(n, p)$. Therefore $\boldsymbol{AB'}$ has a lower rank than $\boldsymbol{X}$ and we thus have that the true ratings are explained by a small set of latent factors.

In the estimation process, we fit the low-rank matrix $\boldsymbol{AB'}$ to $\boldsymbol{X}$ by minimizing some objective function. The confidence we have in each observation is included in the model by putting weights on each observation's contribution to the objective function. These weights are collected in $n \times p$ matrix $\boldsymbol{W}$. We will elaborate more on the structure and choice of $\boldsymbol{W}$ in Section 2.4.

The general format of the objective function is given by:

$$L(\boldsymbol{A}, \boldsymbol{B} | \boldsymbol{X}, \boldsymbol{W}) = C(\boldsymbol{A}, \boldsymbol{B} | \boldsymbol{X}, \boldsymbol{W}) + \lambda R(\boldsymbol{A}, \boldsymbol{B} | \boldsymbol{W}), \tag{1}$$

where $C(\cdot)$ is a loss function that measures how well our estimate $\boldsymbol{AB'}$ reflects the true data matrix $\boldsymbol{X}$, $R(\cdot)$ is a regularization term that promotes a low rank solution and $\lambda \in \mathbb{R}_+$ controls the importance of the regularization term. In general, λ needs to be selected through cross-validation.

Suppose $\widehat{\boldsymbol{A}}$ and $\widehat{\boldsymbol{B}}$ are associated with the minimum of (1). Let the rows of $\widehat{\boldsymbol{A}}$ and $\widehat{\boldsymbol{B}}$ be denoted by column vectors $\hat{\boldsymbol{a}}_i$ and $\hat{\boldsymbol{b}}_j$; these are the user and item vectors in the latent feature space. Now, each entry of $\widehat{\boldsymbol{AB}}'$ will be given by the dot product $\hat{\boldsymbol{a}}_i' \hat{\boldsymbol{b}}_j$. The value of this dot-product is assumed to represent the underlying

preference of a user for an item. These preferences eventually express themselves in $\boldsymbol{X}$, the binary implicit feedback matrix. From the predicted preferences, we can construct our recommendations.

The notations introduced in this section will be used throughout the paper and, where necessary, additional notation will be provided. In order to facilitate notation, we assume throughout this paper that $n \geq p$, even though the results hold for the case that $n < p$ as well.

2.3 Implicit Feedback Models

There are many possible choices for the cost function $C(\cdot)$ and regularization $R(\cdot)$, but not all are equally well suited to the implicit feedback context. For explicit feedback, the predominant cost function in literature is the quadratic loss function over the observed entries of the matrix. As a regularization term for convex optimization methods, any candidate that is a convex relaxation of the rank constraint can be used. In literature, the Nuclear Norm (NN, also known as trace norm and Ky Fan norm) is often used as regularization term since this the "best" convex approximation of the rank (Candes and Recht, 2008; Fazel, 2002; Mazumder et al., 2010). The NN is defined as the sum of singular values and the rationale behind its use is that its minimum is likely to be obtained if many of the singular values are set to zero, thus resulting in a low rank solution. Let $\Omega := \{(i,j) : x_{ij} \quad \text{is observed}\}$ denote the set of observed entries. The model to be minimized then becomes (Mazumder et al., 2010):

$$\underset{\boldsymbol{A},\boldsymbol{B}}{\text{minimize}} \quad f_\lambda(\boldsymbol{A},\boldsymbol{B}) = \frac{1}{2}\sum_{(i,j)\in\Omega}(x_{ij} - \boldsymbol{a}_i'\boldsymbol{b}_j)^2 + \lambda||\boldsymbol{A}\boldsymbol{B}'||_*. \tag{2}$$

where $||\cdot||_*$ denotes the NN.

For Implicit feedback models, we cannot follow the approach of only including observed values in the analysis. Namely, we need to include the information of all the data points (Case 1 and Case 2, see Section 2.2) in our model. In addition, we need to include the confidence we have in the observation through weights. A simple modification of (2) that suits the implicit feedback case is to change the cost function into weighted squared loss (Hu et al., 2008):

$$\underset{\boldsymbol{A},\boldsymbol{B}}{\text{minimize}} \quad f_\lambda(\boldsymbol{A},\boldsymbol{B}) = \frac{1}{2}\left[\sum_{i,j} w_{ij}(x_{ij} - \boldsymbol{a}_i'\boldsymbol{b}_j)^2\right] + \lambda||\boldsymbol{A}\boldsymbol{B}'||_*. \tag{3}$$

Note that (2) can be seen as a special case of this function, where we have set $w_{ij} = 1$ for $x_{ij} \in \Omega$ and zero elsewhere.

In Section 3.3 we will illustrate that the optimization of (3) can be slow in the presence of extreme weights. In Section 3.4 this problem is solved by substituting

the NN for the Weighted Nuclear Norm (WNN). The WNN is also a convex regularization term and it is defined in Appendix A.1. We leave the motivation for this model to Section 3.4 since it requires in-depth knowledge on the optimization procedure.

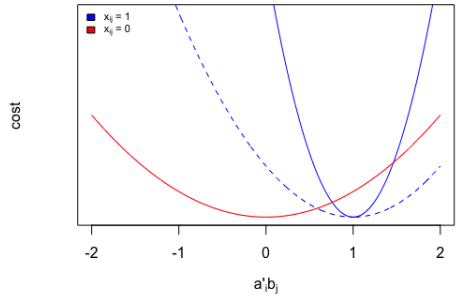


Figure 1: Quadratic loss functions for $x_{ij} = 1$ and $x_{ij} = 0$. The dashed line indicates a loss function for $x_{ij} = 1$ with lower weight.

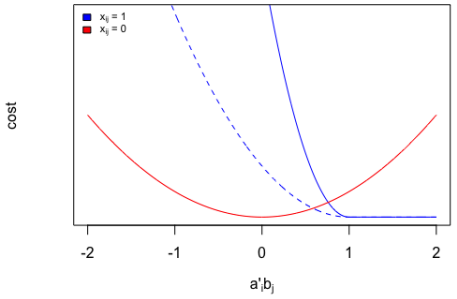


Figure 2: One-sided squared hinge loss functions for $x_{ij} = 1$ and $x_{ij} = 0$. The dashed line indicates a loss function for $x_{ij} = 1$ with lower weight.

In Figure 1, we see the quadratic loss function penalizes over- and underestimations equally hard. The weights determine the steepness of the curves. We argue that quadratic loss is not the most appropriate cost function for implicit data and a one-sided squared hinge loss function, as in Figure 2 is more appropriate.

If the numerical value of $\hat{\boldsymbol{a}}_i'\hat{\boldsymbol{b}}_j$ reveals the underlying preference of a person, we do not want to push estimates towards $x_{ij} = 1$ if a higher predicted estimate is more appropriate. That is, assume that the preference follows a linear scale. Let us associate negative values with negative preferences, a zero value with neutral preferences and a positive value with positive preferences. Then, if we have observed feedback (Case 1), we want to promote higher values, which do not necessarily need to be equal to 1. We still want to apply a squared loss for values if $x_{ij} = 0$. This holds since we are unaware of the underlying reason, i.e. we do not know whether it is unobserved because of dislike or because the user was simply unaware of the product. If we translate these losses in our objective function, we have:

$$\begin{aligned} \underset{\boldsymbol{A},\boldsymbol{B}}{\text{minimize}} \quad f_\lambda(\boldsymbol{A},\boldsymbol{B}) = \frac{1}{2}[& \sum_{(i,j)\in\Omega} w_{ij} \max(0, 1 - \boldsymbol{a}_i'\boldsymbol{b}_j)^2 \\ & + \sum_{(i,j)\notin\Omega} w_{ij}(\boldsymbol{a}_i'\boldsymbol{b}_j)^2] + \lambda R(\boldsymbol{A},\boldsymbol{B}|\boldsymbol{W}) \end{aligned} \tag{4}$$

2.4 Incorporating Information by Weighting Matrix $\boldsymbol{W}$

The weights in the loss function express the confidence we have in the observation and they incorporate all sorts of information into the model, e.g. temporal information or information on the customer segment.

As mentioned in Section 2.2, we want to have a (relatively) large positive weight for observations in case 1 and a small positive weight for observations in case 2. Therefore, we restrict our weights as:

$$w_{ij} = \begin{cases} 1 + \psi_{ij} & \text{if } x_{ij} = 1 \\ 1 & \text{if } x_{ij} = 0 \end{cases}, \tag{5}$$

where ψ_{ij} is required to be positive and depends on the numerical value of the implicit feedback.

This structure of the weighting matrix is still very flexible and allows us to incorporate many forms of information in the model. Note that $\boldsymbol{W} = \boldsymbol{J} + \boldsymbol{\Psi}$, if ψ_{ij} are the elements of $n \times p$ matrix $\boldsymbol{\Psi}$, with the same sparsity structure as $\boldsymbol{X}$ and $\boldsymbol{J}$ is a $n \times p$ matrix of ones. By including weights for $x_{ij} = 0$, we also prevent the trivial low-rank solution of $\widehat{\boldsymbol{AB}}' = \boldsymbol{J} = \mathbf{1}_{(n\times 1)}\mathbf{1}'_{(p\times 1)}$, where $\mathbf{1}$ is a vector of ones.

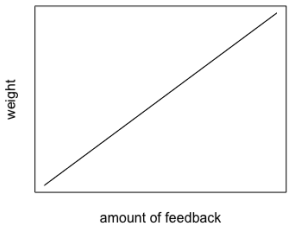
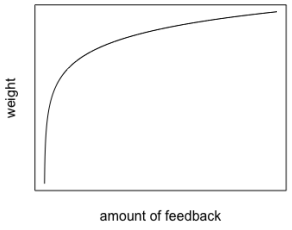
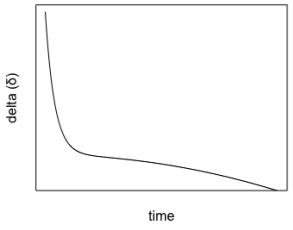
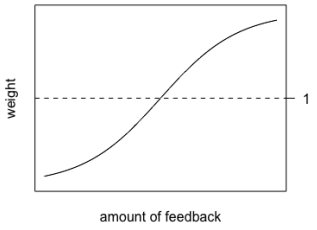
Let us define the input data as $\tau_{ijt} \in \mathbb{R}_+$, where partial observations are allowed (for example: half-played music streams) and t denotes the time-stamp corresponding to the observation. Let T_{ij} denote the set of time-stamps corresponding to user i and item j. Also, let α denote a scaling that can be set by the researcher.

The choice of a proper weighting scheme is very dependent on the context in which the recommender is used. In Table 1, we provide four examples and the selection of the final scheme should be done on the basis of managerial input as well as testing. In this process, two aspects need to be considered.

As a first aspect, one needs to decide how the feedback is translated into the weights. For example, the researcher must decide whether a constant or decreasing marginal effect of feedback on the weight is more appropriate in the context (i.e. simple sum or logarithmic weighing scheme). In addition, the researcher could incorporate other contextual information in the model through weights on the feedback points (i.e. the time decay - weighted scheme, which is motivated by the assumption that taste changes over time). Another option would be to normalize the weights, this has the effect of reducing the variance among users, however, it could also lead to over/underestimation of the confidence we have in a observation.

A second aspect that needs to be considered is the scaling parameter (α). The researcher needs to tune this parameter to an appropriate value. The optimal value strongly depends on the selected weighing scheme, the distribution of the data and the type of input data. One should look over a grid of values of α and select the value for which the model shows the best out-of-sample performance.

Table 1: Some examples of allowed weighting schemes

Weighting Scheme	Illustration	Explanation
Simple sum $\psi_{ij} = \alpha \sum_{t \in T_{ij}} \tau_{ijt}$		This is a simple weighing scheme where the importance increases by α with the numerical value of the feedback. ψ_{ij} actually expresses the confidence we have in the positive feedback.
Logarithmic $\psi_{ij} = \alpha \log(1 + \sum_{t \in T_{ij}} \tau_{ijt})$		The weights increase logarithmically, which reduces the relative effect of extreme weights.
Time decay - weighted $\psi_{ij} = \alpha \sum_{t \in T_{ij}} \delta_t \tau_{ijt}$		Here, we can use a time-decay function that, for instance, puts lower importance on the older feedback points within a simple sum through δ_t.
Product Normalized $\psi_{ij} = \alpha \frac{\sum_{t \in T_{ij}} \tau_{ijt}}{\text{mean}_{j \in J}\left(\sum_{t \in T_{ij}} \tau_{ijt}\right)}$		Normalized weights help to reduce the effect of extreme weights and center the average weight on 1. The weight is dependent on the average feedback from a user.

3 Optimization

3.1 `softImpute` Algorithm

Almost all algorithms mentioned in Section 2.1 use either alternating least squares (ALS) or stochastic gradient descent (SGD) techniques to optimize their respective loss function. These methods can be seen as the industry standard, they tend to show good performance and are widely applied for both implicit as well as explicit feedback models.

Mazumder et al. (2010) designed an efficient convex spectral regularization algorithm to optimize (2). Their `softImpute` algorithm outperforms state-of-the-art SGD techniques in terms of speed. `softImpute` not necessarily outperforms ALS type algorithms in terms of speed (Hastie et al., 2015), but convex optimization methods typically require less observations then ALS (Jain et al., 2013). Therefore, the `softImpute` algorithm seems like an attractive alternative to the existing methods.

However, until now, the method only works in the context of explicit feedback. Mazumder et al. (2010) cleverly exploits the problem structure for explicit feedback. From (2) we see that they only include the observed values in their analysis. Because of this, they can efficiently compute the solution. Let $\boldsymbol{W}^\star$ be a weighting matrix with 1 if $x_{ij} \in \Omega$ and 0 elsewhere and let $\odot$ denote element-wise multiplication, then Mazumder et al. (2010) show that their algorithm needs to iterate between two steps:

1. Replace the missing values in $\boldsymbol{X}$ with the corresponding values from the current estimate $\widehat{\boldsymbol{AB}'}$:

$$\widehat{\boldsymbol{X}} \leftarrow [\boldsymbol{W}^\star \odot \boldsymbol{X} - \boldsymbol{W}^\star \odot (\widehat{\boldsymbol{AB}}')] + \widehat{\boldsymbol{AB}}'. \tag{6}$$

2. Update $\widehat{\boldsymbol{AB}}'$ by means of a soft-thresholded Singular Value Decomposition (SVD) of $\widehat{\boldsymbol{X}}$:.

$$\widehat{\boldsymbol{X}} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}' \tag{7}$$

$$\widehat{\boldsymbol{AB}}' \leftarrow \boldsymbol{U}S_\lambda(\boldsymbol{D})\boldsymbol{V}', \tag{8}$$

   where the soft-thresholding operator, $S_\lambda(\cdot)$, is an element-wise operator that replaces each element on the diagonal of $\boldsymbol{D}$ with $\max(d_{ii} - \lambda, 0)$.

The SVD that needs to be calculated in (7) provides a computational bottleneck, especially if we scale it to large matrices. Fortunately, Mazumder et al. (2010) are able to overcome this bottleneck by exploiting the *sparse-plus-low-rank* structure

of $\widehat{\boldsymbol{X}}$, which is a result of only including observations in Ω. Sparse-plus-low-rank matrices are inexpensive to store and compute.

In addition, they compute a reduced rank SVD at each step to speed up the calculations even further. At the time of writing the article, there were already methods available that can do this while exploiting the sparse plus low rank structure. However, the same authors developed a new alternating subspace type algorithm in Hastie et al. (2015), that achieves even a bigger speed up because it exploits warm starts.

These warm starts can be exploited because the algorithm uses a regularization path of λ's. In order find the value for λ for which it performs best, it first uses large λ's that cause the eventual solution to be of low rank. These low-rank solutions can be used as warm starts for the next run of the algorithm with a lower λ.

It is because of these two tricks that the `softImpute` algorithm is very efficient for large datasets and shows superior performance to state-of-the-art SGD and ALS techniques. In Mazumder et al. (2010), they show that the algorithm can estimate the Netflix dataset in under 4 hours. The possibility to apply the algorithm in batch on such big datasets is one of the most attractive properties of the algorithm.

At first glance, one might expect that we cannot use these tricks in our implicit case. From Section 2.3, we know that we need to include all data points in our analysis. Nonetheless, we are able to combine the sparsity in $\boldsymbol{X}$ together with the structure of $\boldsymbol{W}$ as defined in (5) in order to apply a similar trick as Mazumder et al. (2010) in our algorithms. In the next sections, we will show this by constructing three algorithms from the ground up by Minimization by Majorization.

3.2 Minimization by Majorization (MM)

Minimization by majorization algorithms exploit the convex structure of an objective function in order to guarantee the descent in each iteration. This type of algorithm is generally known as MM-algorithm and its origins can be traced back to 1970 (Ortega and Rheinboldt, 1970).
When we have a convex objective function $f(\boldsymbol{\theta})$ that we want to minimize, the MM-algorithms work by finding a surrogate function, the majorizing function $g(\boldsymbol{\theta}|\boldsymbol{\theta}^{(o)})$, that satisfies:

1. $f(\boldsymbol{\theta}^{(o)}) = g(\boldsymbol{\theta}^{(o)}|\boldsymbol{\theta}^{(o)})$,
2. $f(\boldsymbol{\theta}) \leq g(\boldsymbol{\theta}|\boldsymbol{\theta}^{(o)})$,

where $\boldsymbol{\theta}^{(o)}$ denotes the current estimate for parameter $\boldsymbol{\theta}$. Hence, $g(\boldsymbol{\theta}|\boldsymbol{\theta}^{(o)})$ touches $f(\boldsymbol{\theta})$ in $\boldsymbol{\theta}^{(o)}$ and lies above or on the function for other values of $\boldsymbol{\theta}$. The algorithm works by minimizing $g(\boldsymbol{\theta}|\boldsymbol{\theta}^{(o)})$ instead of $f(\boldsymbol{\theta})$. Note that it is therefore convenient to have $g(\boldsymbol{\theta}|\boldsymbol{\theta}^{(o)})$ in a simple form, since its minimum $\boldsymbol{\theta}_{\min}$ will be easy to find.

It is easy to show that descent is guaranteed at each iteration. Since for $\boldsymbol{\theta}_{\min} = \text{argmin } g(\boldsymbol{\theta}|\boldsymbol{\theta}^{(o)})$, we must have that $f(\boldsymbol{\theta}_{\min}) \leq g(\boldsymbol{\theta}_{\min}|\boldsymbol{\theta}^{(o)}) \leq g(\boldsymbol{\theta}^{(o)}|\boldsymbol{\theta}^{(o)}) = f(\boldsymbol{\theta}^{(o)})$. Hence, each update improves f until convergence.

In the next three sections, we will derive the closed form expressions of the updates for three different objective functions. Afterwards, we will elaborate on the estimation procedure and computational complexity of all three algorithms. All three algorithms are able to efficiently complete big matrices with implicit data by applying the similar tricks as in Mazumder et al. (2010).

3.3 Algorithm 1: `Weighted-softImpute (WSI)`

We first aim to optimize the first implicit data model from Section 2.3. Recall that this model is given by:

$$L_1(\boldsymbol{A}, \boldsymbol{B}) = \frac{1}{2}\left[\sum_{i,j} w_{ij}(x_{ij} - \boldsymbol{a}_i'\boldsymbol{b}_j)^2\right] + \lambda||\boldsymbol{A}\boldsymbol{B}'||_*. \tag{9}$$

Let $\boldsymbol{v} = \text{Vec}(\boldsymbol{A}\boldsymbol{B}')$ be the vectorization of $\boldsymbol{A}\boldsymbol{B}'$. Here $\boldsymbol{v}$ is a column vector with $v_{ij} = \boldsymbol{a}_i'\boldsymbol{b}_j$. Similarly, let $\boldsymbol{x} = \text{Vec}(\boldsymbol{X})$. Let $\boldsymbol{D}_{\boldsymbol{W}} = \text{diag}(\text{Vec}(\boldsymbol{W}))$ be a matrix with the weights on the diagonal.

If m is the maximum of these weights, we must have that $\boldsymbol{D}_{\boldsymbol{W}} - m\boldsymbol{I}$ is negative semi-definite. We can use this fact in the derivation of our majorizing function:

$$\begin{aligned}(\boldsymbol{v} - \boldsymbol{v}^{(o)})'(\boldsymbol{D}_{\boldsymbol{W}} - m\boldsymbol{I})(\boldsymbol{v} - \boldsymbol{v}^{(o)}) \leq 0,& \\ \boldsymbol{v}'\boldsymbol{D}_{\boldsymbol{W}}\boldsymbol{v} \leq m\boldsymbol{v}'\boldsymbol{v} - 2m\boldsymbol{v}'&\left[\boldsymbol{v}^{(o)} - m^{-1}\boldsymbol{D}_{\boldsymbol{W}}\boldsymbol{v}^{(o)}\right] \\ &+ m\boldsymbol{v}^{(o)\prime}\boldsymbol{v}^{(o)} - \boldsymbol{v}^{(o)\prime}\boldsymbol{D}_{\boldsymbol{W}}\boldsymbol{v}^{(o)}\end{aligned} \tag{10}$$

Plugging (10) in (9), yields our majorizing function:

$$\begin{aligned}L_1(\boldsymbol{A}, \boldsymbol{B}) &= \frac{1}{2}\left[\boldsymbol{x}'\boldsymbol{D}_{\boldsymbol{W}}\boldsymbol{x} - 2\boldsymbol{v}'\boldsymbol{D}_{\boldsymbol{W}}\boldsymbol{x} + \boldsymbol{v}'\boldsymbol{D}_{\boldsymbol{W}}\boldsymbol{v}\right] + \lambda||\boldsymbol{A}\boldsymbol{B}'||_* \\ &\leq \frac{1}{2}[\boldsymbol{x}'\boldsymbol{D}_{\boldsymbol{W}}\boldsymbol{x} + m\boldsymbol{v}'\boldsymbol{v} - 2m\boldsymbol{v}'(\boldsymbol{v}^{(o)} - m^{-1}\boldsymbol{D}_{\boldsymbol{W}}\boldsymbol{v}^{(o)} + m^{-1}d\boldsymbol{D}_{\boldsymbol{W}}\boldsymbol{x}) \\ &\quad + m\boldsymbol{v}^{(o)\prime}\boldsymbol{v}^{(o)} - \boldsymbol{v}^{(o)\prime}\boldsymbol{D}_{\boldsymbol{W}}\boldsymbol{v}^{(o)}] + \lambda||\boldsymbol{A}\boldsymbol{B}'||_* \\ &= \frac{1}{2}\sum_{i,j}\left(mv_{ij}^2 - 2mv_{ij}r_{ij} + mr_{ij}^2\right) + c + \lambda||\boldsymbol{A}\boldsymbol{B}'||_* \\ &= \frac{m}{2}\text{tr}(\boldsymbol{R} - \boldsymbol{A}\boldsymbol{B}')'(\boldsymbol{R} - \boldsymbol{A}\boldsymbol{B}') + c + \lambda||\boldsymbol{A}\boldsymbol{B}'||_* \\ &= g_1(\boldsymbol{A}, \boldsymbol{B}|\boldsymbol{A}^{(o)}, \boldsymbol{B}^{(o)}),\end{aligned} \tag{11}$$

where

$$r_{ij} = \frac{w_{ij}}{m} x_{ij} - \frac{w_{ij}}{m} v_{ij}^{(o)} + v_{ij}^{(o)} \tag{12}$$

and $c = \frac{1}{2}\sum_{i,j}(w_{ij}x_{ij}^2 - mr_{ij}^2 + mv_{ij}^{(o)2} - w_{ij}v_{ij}^{(o)2})$.

In order to update $\boldsymbol{AB'}$ we need to find an analytical expression for the minimum of (11). Lets define the SVD's $\boldsymbol{R} = \boldsymbol{P\Phi Q'}$ and $\boldsymbol{AB'} = \boldsymbol{U\Sigma V'}$. We now need to minimize the function with respect to $\boldsymbol{U}$, $\boldsymbol{\Sigma}$ and $\boldsymbol{V}$. By substituting $\boldsymbol{R}$ and $\boldsymbol{AB'}$, we can rewrite (11) as:

$$\frac{m}{2}\left(\text{tr}(\boldsymbol{\Phi}^2) - 2\text{tr}(\boldsymbol{Q\Phi P'U\Sigma V'}) + \text{tr}(\boldsymbol{\Sigma}^2)\right) + \lambda||\boldsymbol{\Sigma}||_* + c. \tag{13}$$

For updating $\boldsymbol{U}$ and $\boldsymbol{V}$, it suffices to minimize $-2\text{tr}(\boldsymbol{Q\Phi P'U\Sigma V'})$, or, similarly, maximize $\text{tr}(\boldsymbol{Q\Phi P'U\Sigma V'})$. A maximum of this trace function is given by the Kristof upper bound (Kristof, 1970):

$$\begin{aligned}\text{tr}(\boldsymbol{Q\Phi P'U\Sigma V'}) &= \text{tr}(\boldsymbol{\Phi}(\boldsymbol{P'U})\boldsymbol{\Sigma}(\boldsymbol{V'Q})) \\ &\leq \text{tr}(\boldsymbol{\Phi I\Sigma I})\end{aligned} \tag{14}$$

Hence, a maximum is attained if $\boldsymbol{U} = \boldsymbol{P}$ and $\boldsymbol{V} = \boldsymbol{Q}$.
In order to update $\boldsymbol{\Sigma}$, we rewrite (13) with these values plugged in:

$$\begin{aligned}&\frac{m}{2}\left[\sum_{j=1}^{p}\phi_{jj}^2 + \sum_{j=1}^{p}\sigma_{jj}^2 - 2\sum_{j=1}^{p}\phi_{jj}\sigma_{jj}\right] + \lambda\sum_{j=1}^{p}\sigma_{jj} \\ &= \sum_{j=1}^{p}\left[\frac{m}{2}(\phi_{jj} - \sigma_{jj})^2 + \lambda\sigma_{jj}\right].\end{aligned} \tag{15}$$

By definition, we must have that $\sigma_{jj} \geq 0$. Therefore the updates are givenby:

$$\sigma_{jj} = \max(0, \phi_{jj} - \frac{\lambda}{m}), \tag{16}$$

for each $j = 1, \ldots, p$. Or, put differently, $\boldsymbol{\Sigma} = S_{\lambda/m}(\boldsymbol{\Phi})$, where $S_{\lambda/m}(\cdot)$ is a threshold operator that applies (16) element-wise to $\boldsymbol{\Phi}$. Hence, the total update for each iteration is:

$$\boldsymbol{AB'} = \boldsymbol{P}S_{\lambda/m}(\boldsymbol{\Phi})\boldsymbol{Q'}. \tag{17}$$

For each update, we thus need to first calculate a thresholded singular value decomposition of $\boldsymbol{R}$. This is the most computationally intensive part of the calculation. We aim to overcome this bottleneck by exploiting the same *sparse-plus-low-rank* structure as in Mazumder et al. (2010). Let us rewrite (12) in matrix form

as:

$$\begin{aligned}\boldsymbol{R} &= [m^{-1}\boldsymbol{W} \odot \boldsymbol{X} - m^{-1}\boldsymbol{W} \odot (\boldsymbol{A}^{(o)}\boldsymbol{B}^{(o)\prime})] + \boldsymbol{A}^{(o)}\boldsymbol{B}^{(o)\prime} \\ &= [m^{-1}(\boldsymbol{J} + \boldsymbol{\Psi}) \odot \boldsymbol{X} - m^{-1}(\boldsymbol{J} + \boldsymbol{\Psi}) \odot (\boldsymbol{A}^{(o)}\boldsymbol{B}^{(o)\prime})] + \boldsymbol{A}^{(o)}\boldsymbol{B}^{(o)\prime} \\ &= [m^{-1}(\boldsymbol{J} + \boldsymbol{\Psi}) \odot \boldsymbol{X} - m^{-1}\boldsymbol{\Psi} \odot (\boldsymbol{A}^{(o)}\boldsymbol{B}^{(o)\prime})] + \frac{m-1}{m}\boldsymbol{A}^{(o)}\boldsymbol{B}^{(o)\prime}. \end{aligned} \tag{18}$$

Because we have that the entries of $\boldsymbol{X}$ are zero if the entries of $\boldsymbol{\Psi}$ are zero as well, we have that the part in brackets on the left is sparse. Since the part on the right is low rank, we can store $\boldsymbol{R}$ as a sparse-plus-low-rank matrix and apply the methods developed in Hastie et al. (2015) to efficiently calculate the thresholded singular value decomposition.

This algorithm tries to fit $\boldsymbol{AB}'$ as close to $\boldsymbol{R}'$ as possible. But, if $w_{ij} \ll m$, we can see from (12) that $\boldsymbol{R}$ consists mostly of the old estimate and only for $\frac{w_{ij}}{m}$ of the true data. As a result, $\boldsymbol{AB}'$ will be fitted for a large part to the old estimate. This leads to slow convergence if there is a lot of deviation between the largest w_{ij} and the other weights. In the next section we will try to overcome this problem and allow for better convergence.

3.4 Algorithm 2: `Row-weighted-softImpute (RWSI)`

In Groenen et al. (2003), the authors propose several majorization algorithms that optimize a weighted least squares loss function without a regularization term. They propose to majorize each row separately with its row maximum in order to reduce the effect found in the previous section. Their study shows a significant speed up as well as more accurate results. In this section, we try to adopt their approach in our regularized case.

The optimization of the majorizing function in Groenen et al. (2003) involves a Generalized Singular Value Decomposition (GSVD) (Greenacre, 1984; Groenen et al., 2003; Takane and Shibayama, 1991). Because of this, we encounter a problem with the Nuclear Norm. That is, the GSVD involves generalized singular values, but the NN is defined as the sum of the regular singular values. Ultimately, this implies that in the equivalent of (15) there will be both singular values as well as generalized singular values of $\boldsymbol{AB}'$. Hence, we cannot find the optimal value of our estimate.

We solve this problem by substituting the NN by the Weighted Nuclear Norm (WNN) as defined in Angst et al. (2011), where they call it the Generalized Trace Norm. For a comprehensive definition, we refer to their paper and Appendix A.1.

Let $\boldsymbol{D}_r$ and $\boldsymbol{D}_c$ be positive semi-definite matrices of sizes $n \times n$ and $p \times p$. If we write $\boldsymbol{D}_r = \boldsymbol{V}_r\boldsymbol{\Lambda}_r\boldsymbol{V}_r$ and $\boldsymbol{D}_c = \boldsymbol{V}_c\boldsymbol{\Lambda}_c\boldsymbol{V}_c$ to be the eigen-decompositions, then the

WNN ($||\cdot||_{*(\boldsymbol{D}_r,\boldsymbol{D}_c)}$) can be related to the NN as:

$$||\boldsymbol{X}||_{*(\boldsymbol{D}_r,\boldsymbol{D}_c)} = ||\boldsymbol{V}_r\boldsymbol{\Lambda}_r^{1/2}\boldsymbol{X}\boldsymbol{\Lambda}_c^{1/2}\boldsymbol{V}_c'||_* = ||\boldsymbol{C}_r\boldsymbol{X}\boldsymbol{C}_c'||_*, \tag{19}$$

where $\boldsymbol{C}_r = \boldsymbol{V}_r\boldsymbol{\Lambda}_r^{1/2}$ and $\boldsymbol{C}_c = \boldsymbol{V}_c\boldsymbol{\Lambda}_c^{1/2}$ (See Appendix A.1). The advantages of this norm will become clear during the derivation of our second algorithm.

Using this WNN, we have a new loss function:

$$L_2(\boldsymbol{A},\boldsymbol{B}) = \frac{1}{2}\left[\sum_{i,j} w_{ij}(x_{ij} - \boldsymbol{a}_i'\boldsymbol{b}_j)^2\right] + \lambda||\boldsymbol{A}\boldsymbol{B}'||_{*(\boldsymbol{D_m},\boldsymbol{I})}, \tag{20}$$

where $m_i = \max_j w_{ij}$ and $\boldsymbol{D_m}$ is defined as a diagonal matrix with the m_i's on the diagonal, hence it is a $n \times n$ positive definite matrix.

Let the rows of our estimate be given in column vector $\boldsymbol{v}_i = \boldsymbol{B}\boldsymbol{a}_i$, let the row's of $\boldsymbol{X}$ be denoted by column vector $\boldsymbol{x}_i$ and let $\boldsymbol{D}_{\boldsymbol{W}_i}$ be defined as a diagonal matrix with the weights corresponding to $\boldsymbol{x}_i$. Now, following the idea from Groenen et al. (2003), we change majorizing formula (10) to:

$$\begin{aligned}(\boldsymbol{v}_i - \boldsymbol{v}_i^{(o)})'(\boldsymbol{D}_{\boldsymbol{W}_i} - m_i\boldsymbol{I})(\boldsymbol{v}_i - \boldsymbol{v}_i^{(o)}) &\leq 0 \\ \boldsymbol{v}_i'\boldsymbol{D}_{\boldsymbol{W}_i}\boldsymbol{v}_i &\leq m_i\boldsymbol{v}_i'\boldsymbol{v}_i - 2m_i\boldsymbol{v}_i'\left[\boldsymbol{v}_i^{(o)} - m_i^{-1}\boldsymbol{D}_{\boldsymbol{W}_i}\boldsymbol{v}_i^{(o)}\right] \\ &\quad + m_i\boldsymbol{v}_i^{(o)\prime}\boldsymbol{v}_i^{(o)} - \boldsymbol{v}_i^{(o)\prime}\boldsymbol{D}_{\boldsymbol{W}_i}\boldsymbol{v}_i^{(o)}.\end{aligned} \tag{21}$$

We can use this to majorize loss function (20) as:

$$\begin{aligned}L_2(\boldsymbol{A},\boldsymbol{B}) &= \frac{1}{2}\sum_i\left[\boldsymbol{x}_i'\boldsymbol{D}_{\boldsymbol{W}_i}\boldsymbol{x}_i - 2\boldsymbol{v}_i'\boldsymbol{D}_{\boldsymbol{W}_i}\boldsymbol{x}_i + \boldsymbol{v}_i'\boldsymbol{D}_{\boldsymbol{W}_i}\boldsymbol{v}_i\right] + \lambda||\boldsymbol{A}\boldsymbol{B}'||_{*(\boldsymbol{D_m},\boldsymbol{I})} \\ &\leq \frac{1}{2}\sum_i m_i\sum_j\left(v_{ij}^2 - 2v_{ij}r_{ij} + r_{ij}^2\right) + c + \lambda||\boldsymbol{A}\boldsymbol{B}'||_{*(\boldsymbol{D_m},\boldsymbol{I})} \\ &= \frac{1}{2}\mathrm{tr}(\boldsymbol{R} - \boldsymbol{A}\boldsymbol{B}')'\boldsymbol{D_m}(\boldsymbol{R} - \boldsymbol{A}\boldsymbol{B}') + c + \lambda||\boldsymbol{A}\boldsymbol{B}'||_{*(\boldsymbol{D_m},\boldsymbol{I})} \\ &= g_2(\boldsymbol{A},\boldsymbol{B}|\boldsymbol{A}^{(o)},\boldsymbol{B}^{(o)}),\end{aligned} \tag{22}$$

where $c = \frac{1}{2}\sum_{i,j}\left(w_{ij}x_{ij}^2 - m_ir_{ij}^2 + m_iv_{ij}^{(o)2} - w_{ij}v_{ij}^{(o)2}\right)$. Also, we have:

$$r_{ij} = \frac{w_{ij}}{m_i}x_{ij} + v_{ij}^{(o)} - \frac{w_{ij}}{m_i}v_{ij}^{(o)}, \tag{23}$$

from which we can deduce that the problem of possible slow convergence is likely to be reduced because of the row-weights.

In order to optimize this function, let $\boldsymbol{D}_{\boldsymbol{m}}^{1/2}\boldsymbol{R} = \boldsymbol{Q}\boldsymbol{\Phi}\boldsymbol{P}'$ be a regular SVD, then we set $\boldsymbol{R} = \widetilde{\boldsymbol{Q}}\widetilde{\boldsymbol{\Phi}}\widetilde{\boldsymbol{P}}'$ as the GSVD where $\widetilde{\boldsymbol{Q}} = \boldsymbol{D}_{\boldsymbol{m}}^{-1/2}\boldsymbol{Q}$, $\widetilde{\boldsymbol{\Phi}} = \boldsymbol{\Phi}$ and $\widetilde{\boldsymbol{P}} = \boldsymbol{P}$. We have $\widetilde{\boldsymbol{Q}}'\boldsymbol{D}_{\boldsymbol{m}}\widetilde{\boldsymbol{Q}} = \boldsymbol{I}$ and $\widetilde{\boldsymbol{P}}'\widetilde{\boldsymbol{P}} = \boldsymbol{I}$. Let $\boldsymbol{A}\boldsymbol{B}' = \widetilde{\boldsymbol{U}}\widetilde{\boldsymbol{\Sigma}}\widetilde{\boldsymbol{V}}'$ be another GSVD with $\widetilde{\boldsymbol{U}}'\boldsymbol{D}_{\boldsymbol{m}}\widetilde{\boldsymbol{U}} = \boldsymbol{I}$ and $\widetilde{\boldsymbol{V}}'\widetilde{\boldsymbol{V}} = \boldsymbol{I}$. Using this, we can rewrite (22):

$$\begin{aligned} g_2(\boldsymbol{A},\boldsymbol{B}|\boldsymbol{A}^{(o)},\boldsymbol{B}^{(o)}) &= \frac{1}{2}\mathrm{tr}(\boldsymbol{D}_{\boldsymbol{m}}^{1/2}\boldsymbol{R} - \boldsymbol{D}_{\boldsymbol{m}}^{1/2}\boldsymbol{A}\boldsymbol{B}')'(\boldsymbol{D}_{\boldsymbol{m}}^{1/2}\boldsymbol{R} - \boldsymbol{D}_{\boldsymbol{m}}^{1/2}\boldsymbol{A}\boldsymbol{B}') \\ &\quad + c + \lambda||\boldsymbol{A}\boldsymbol{B}'||_{*(\boldsymbol{D}_{\boldsymbol{m}},\boldsymbol{I})} \\ &= \frac{1}{2}[\mathrm{tr}(\widetilde{\boldsymbol{\Phi}}^2) - 2\mathrm{tr}(\widetilde{\boldsymbol{P}}\widetilde{\boldsymbol{\Phi}}\widetilde{\boldsymbol{Q}}'\boldsymbol{D}_{\boldsymbol{m}}\widetilde{\boldsymbol{U}}\widetilde{\boldsymbol{\Sigma}}\widetilde{\boldsymbol{V}}') + \mathrm{tr}(\widetilde{\boldsymbol{\Sigma}})] + c + \lambda||\boldsymbol{A}\boldsymbol{B}'||_{*(\boldsymbol{D}_{\boldsymbol{m}},\boldsymbol{I})} \\ &= \frac{1}{2}[\mathrm{tr}(\widetilde{\boldsymbol{\Phi}}^2) - 2\mathrm{tr}(\widetilde{\boldsymbol{P}}\widetilde{\boldsymbol{\Phi}}\widetilde{\boldsymbol{Q}}'\boldsymbol{D}_{\boldsymbol{m}}\widetilde{\boldsymbol{U}}\widetilde{\boldsymbol{\Sigma}}\widetilde{\boldsymbol{V}}') + \mathrm{tr}(\widetilde{\boldsymbol{\Sigma}})] \\ &\quad + c + \lambda||\boldsymbol{D}_{\boldsymbol{m}}^{1/2}\widetilde{\boldsymbol{U}}\widetilde{\boldsymbol{\Sigma}}\widetilde{\boldsymbol{V}}'||_*, \end{aligned} \tag{24}$$

where the last step follows from the fact that $\boldsymbol{D}_{\boldsymbol{m}}$ is a positive definite diagonal matrix, and hence we have that $\boldsymbol{C}_r = \boldsymbol{D}_{\boldsymbol{m}}^{1/2}$ in (19).

We can rewrite the nuclear norm as:

$$\begin{aligned} ||\boldsymbol{D}_{\boldsymbol{m}}^{1/2}\widetilde{\boldsymbol{U}}\widetilde{\boldsymbol{\Sigma}}\widetilde{\boldsymbol{V}}'||_* &= \mathrm{tr}([\widetilde{\boldsymbol{V}}\widetilde{\boldsymbol{\Sigma}}\widetilde{\boldsymbol{U}}'\boldsymbol{D}_{\boldsymbol{m}}\widetilde{\boldsymbol{U}}\widetilde{\boldsymbol{\Sigma}}\widetilde{\boldsymbol{V}}']^{1/2}) \\ &= \mathrm{tr}([\widetilde{\boldsymbol{V}}\widetilde{\boldsymbol{\Sigma}}^2\widetilde{\boldsymbol{V}}']^{1/2}) \\ &= \mathrm{tr}([(\widetilde{\boldsymbol{V}}\widetilde{\boldsymbol{\Sigma}}\widetilde{\boldsymbol{V}}')^2]^{1/2}) = ||\widetilde{\boldsymbol{\Sigma}}||_*. \end{aligned} \tag{25}$$

Thus, we must minimize

$$g_2(\boldsymbol{A},\boldsymbol{B}|\boldsymbol{A}^{(o)},\boldsymbol{B}^{(o)}) = \frac{1}{2}[\mathrm{tr}(\widetilde{\boldsymbol{\Phi}}^2) - 2\mathrm{tr}(\widetilde{\boldsymbol{P}}\widetilde{\boldsymbol{\Phi}}\widetilde{\boldsymbol{Q}}'\boldsymbol{D}_{\boldsymbol{m}}\widetilde{\boldsymbol{U}}\widetilde{\boldsymbol{\Sigma}}\widetilde{\boldsymbol{V}}') + \mathrm{tr}(\widetilde{\boldsymbol{\Sigma}})] + c + ||\widetilde{\boldsymbol{\Sigma}}||_* \tag{26}$$

In order to find updates for $\widetilde{\boldsymbol{U}}$ and $\widetilde{\boldsymbol{V}}$, we proceed as in Section 3.3 and maximize the term $\mathrm{tr}(\widetilde{\boldsymbol{P}}\widetilde{\boldsymbol{\Phi}}\widetilde{\boldsymbol{Q}}'\boldsymbol{D}_{\boldsymbol{m}}\widetilde{\boldsymbol{U}}\widetilde{\boldsymbol{\Sigma}}\widetilde{\boldsymbol{V}}')$. Again, using the logic of the Kristof upper bound of a trace function (Kristof, 1970), we get:

$$\begin{aligned} \mathrm{tr}(\widetilde{\boldsymbol{P}}\widetilde{\boldsymbol{\Phi}}\widetilde{\boldsymbol{Q}}'\boldsymbol{D}_{\boldsymbol{m}}\widetilde{\boldsymbol{U}}\widetilde{\boldsymbol{\Sigma}}\widetilde{\boldsymbol{V}}') &= \mathrm{tr}(\widetilde{\boldsymbol{\Phi}}(\widetilde{\boldsymbol{Q}}'\boldsymbol{D}_{\boldsymbol{m}}\widetilde{\boldsymbol{U}})\widetilde{\boldsymbol{\Sigma}}(\widetilde{\boldsymbol{V}}'\widetilde{\boldsymbol{P}})) \\ &\leq \mathrm{tr}(\widetilde{\boldsymbol{\Phi}}\boldsymbol{I}\widetilde{\boldsymbol{\Sigma}}\boldsymbol{I}) \end{aligned} \tag{27}$$

Hence, a minimum for the function is attained if $\widetilde{\boldsymbol{U}} = \widetilde{\boldsymbol{Q}} = \boldsymbol{D}_{\boldsymbol{m}}^{-1/2}\boldsymbol{Q}$ and $\widehat{\boldsymbol{V}} = \widetilde{\boldsymbol{P}} = \boldsymbol{P}$. In order to update $\widehat{\boldsymbol{\Sigma}}$, we rewrite (26) with these values plugged in:

$$\sum_{j=1}^{p}\left[\frac{1}{2}(\tilde{\phi}_{jj} - \tilde{\sigma}_{jj})^2 + \lambda\tilde{\sigma}_{jj}\right], \tag{28}$$

with $\tilde{\sigma}_{jj} \geq 0$. Hence, we have a minimum at:

$$\tilde{\sigma}_{pp} = \max(0, \tilde{\phi}_{pp} - \lambda), \tag{29}$$

for each $j = 1, \ldots, p$. Or, put differently $\widetilde{\boldsymbol{\Sigma}} = S_\lambda(\widetilde{\boldsymbol{\Phi}}) = S_\lambda(\boldsymbol{\Phi})$ where $S_\lambda(\cdot)$ is an operator that applies (29) element-wise to $\widetilde{\boldsymbol{\Phi}}$ and the second equality holds by definition of $\widetilde{\boldsymbol{\Phi}}$. The total update for each iteration is:

$$\boldsymbol{A}\boldsymbol{B}' = \boldsymbol{D}_{\boldsymbol{m}}^{-1/2}\boldsymbol{Q}S_\lambda(\boldsymbol{\Phi})\boldsymbol{P}'. \tag{30}$$

Because we need to calculate a thresholded singular value decomposition of $\boldsymbol{D}_{\boldsymbol{m}}^{1/2}\boldsymbol{R}$ for each update, we need to be able to do it efficiently. Let us rewrite:

$$\begin{aligned}
\boldsymbol{D}_{\boldsymbol{m}}^{1/2}\boldsymbol{R} &= \boldsymbol{D}_{\boldsymbol{m}}^{1/2}\left([\boldsymbol{D}_{\boldsymbol{m}}^{-1}\boldsymbol{W} \odot \boldsymbol{X} - \boldsymbol{D}_{\boldsymbol{m}}^{-1}\boldsymbol{W} \odot (\boldsymbol{A}^{(o)}\boldsymbol{B}^{(o)\prime})] + \boldsymbol{A}^{(o)}\boldsymbol{B}^{(o)\prime}\right) \\
&= [\boldsymbol{D}_{\boldsymbol{m}}^{-1/2}(\boldsymbol{J}+\boldsymbol{\Psi}) \odot \boldsymbol{X} - \boldsymbol{D}_{\boldsymbol{m}}^{-1/2}(\boldsymbol{J}+\boldsymbol{\Psi}) \odot (\boldsymbol{A}^{(o)}\boldsymbol{B}^{(o)\prime})] + \boldsymbol{D}_{\boldsymbol{m}}^{1/2}\boldsymbol{A}^{(o)}\boldsymbol{B}^{(o)\prime} \\
&= [\boldsymbol{D}_{\boldsymbol{m}}^{-1/2}(\boldsymbol{J}+\boldsymbol{\Psi}) \odot \boldsymbol{X} - \boldsymbol{D}_{\boldsymbol{m}}^{-1/2}(\boldsymbol{\Psi}) \odot (\boldsymbol{A}^{(o)}\boldsymbol{B}^{(o)\prime})] \\
&\quad + (\boldsymbol{D}_{\boldsymbol{m}}^{1/2} - \boldsymbol{D}_{\boldsymbol{m}}^{-1/2})\boldsymbol{A}^{(o)}\boldsymbol{B}^{(o)\prime},
\end{aligned} \tag{31}$$

where we see that we can exploit the sparse-plus-low-rank structure.

3.5 Algorithm 3: `Hinge-Row-Weighted-softImpute (HRWSI)`

Finally, we aim to optimize the last implicit data model from Section 2.3. Recall that this model is given by:

$$L_3(\boldsymbol{A}, \boldsymbol{B}) = \frac{1}{2}\left[\sum_{(i,j)\in\Omega} w_{ij}\max(0, 1 - v_{ij})^2 + \sum_{(i,j)\notin\Omega} w_{ij}(v_{ij})^2\right] + \lambda||\boldsymbol{A}\boldsymbol{B}'||_{*(\boldsymbol{D_m},\boldsymbol{I})}, \tag{32}$$

where we inserted the WNN instead of the NN because we will use the improved row-weighted majorization step from Section 3.4 in this algorithm as well.

We proceed to majorize the first term between brackets by using the approach introduced in Groenen et al. (2008). That is, we notice that $h(v_{ij}) = \max(0, 1-v_{ij})^2$ coincides with $(1-v_{ij})^2$ if $v_{ij}^{(o)} \leq 1$. Hence, we can use this to majorize the first part of $h(v_{ij})$. Then for $v_{ij}^{(o)} > 1$, we want to find a function that has the same curvature as $(1-v_{ij})^2$, but touches at $v_{ij}^{(o)}$. This holds for function $(v_{ij} - v_{ij}^{(o)})^2$. Now, we can express our majorizing function as: $g(\boldsymbol{a}_i, \boldsymbol{b}_j | \boldsymbol{a}_i^{(o)}, \boldsymbol{b}_j^{(o)}) = v_{ij}^2 - 2\beta_{ij}v_{ij} + \gamma_{ij}$, where:

$$\beta_{ij} = \begin{cases} 1 & \text{if } v_{ij}^{(o)} \leq 1 \\ v_{ij}^{(o)} & \text{if } v_{ij}^{(o)} > 1 \end{cases} \tag{33}$$

$$\gamma_{ij} = \begin{cases} 1 & \text{if } v_{ij}^{(o)} \leq 1 \\ (v_{ij}^{(o)})^2 & \text{if } v_{ij}^{(o)} > 1 \end{cases} \tag{34}$$

We can use this majorization step to rewrite our model as follows:

$$\begin{aligned} L_3(\boldsymbol{A}, \boldsymbol{B}) &= \frac{1}{2}\left[\sum_{(i,j)\in\Omega} w_{ij}\max(0, 1-v_{ij})^2 + \sum_{(i,j)\notin\Omega} w_{ij}v_{ij}^2\right] + \lambda||\boldsymbol{AB}'||_{*(\boldsymbol{D_m},\boldsymbol{I})} \\ &\leq \frac{1}{2}\sum_{i,j} w_{ij}\left[x_{ij}(v_{ij}^2 - 2\beta_{ij}v_{ij} + \gamma_{ij}) + (1-x_{ij})v_{ij}^2\right] + \lambda||\boldsymbol{AB}'||_{*(\boldsymbol{D_m},\boldsymbol{I})} \\ &= \frac{1}{2}\sum_{i,j} w_{ij}(h_{ij}-v_{ij})^2 + c_1 + \lambda||\boldsymbol{AB}'||_{*(\boldsymbol{D_m},\boldsymbol{I})}, \end{aligned} \tag{35}$$

where $h_{ij} = x_{ij}\beta_{ij}$ and $c_1 = \frac{1}{2}\sum_{ij} h_{ij}^2 + \gamma_{ij}x_{ij}$.

We notice that (35) is the same as (20). The only difference is the constant term c_1, which is irrelevant for minimization. Hence, we can majorize and eventually solve it using the `RWSI` algorithm:

$$\begin{aligned} L_3(\boldsymbol{A}, \boldsymbol{B}) &\leq \frac{1}{2}\sum_{ij} w_{ij}(h_{ij}-\gamma_{ij})^2 + c_1 + \lambda||\boldsymbol{AB}'||_{*(\boldsymbol{D_m},\boldsymbol{I})} \\ &\leq \frac{1}{2}\mathrm{tr}(\boldsymbol{D_m^{1/2}R} - \boldsymbol{D_m^{1/2}AB}')'(\boldsymbol{D_m^{1/2}R} - \boldsymbol{D_m^{1/2}AB}') \\ &\quad + c_2 + \lambda||\boldsymbol{AB}'||_{*(\boldsymbol{D_m},\boldsymbol{I})} \\ &= g_3(\boldsymbol{A}, \boldsymbol{B}|\boldsymbol{A}^{(o)}, \boldsymbol{B}^{(o)}), \end{aligned} \tag{36}$$

where $c_2 = \frac{1}{2}\sum_{ij}\left(h_{ij}^2 + \gamma_{ij}x_{ij} + w_{ij}h_{ij}^2 - m_i r_{ij} + m_i v_{ij}^{(o)2} - w_{ij}v_{ij}^{(o)2}\right)$ and

$$\begin{aligned} r_{ij} &= \left[1 - \frac{w_{ij}}{m_i}\right] v_{ij}^{(o)} + \frac{w_{ij}}{m_i}h_{ij} \\ &= \left[1 - \frac{1}{m_i}\right] v_{ij}^{(o)} + x_{ij}\left(\frac{w_{ij}}{m_i}\beta_{ij} - \frac{\psi_{ij}}{m_i}v_{ij}^{(o)}\right). \end{aligned} \tag{37}$$

Note that the left part of this equation is low-rank and the right part is sparse, hence we can again exploit the sparse plus low rank nature of matrix $\boldsymbol{R}$. We refer to Section 3.4 for a detailed derivation of the updates.

3.6 Estimation procedure

Even though the three algorithms solve different loss functions, the underlying principles of the `softImpute` algorithm apply for all of them. This section will carefully explain the estimation procedure for all algorithms.

As mentioned in Section 3.1, the algorithms follow a regularization path of λ's and this allows them to exploit warm starts. This regularization path starts at the largest

(generalized) singular value of matrix $\boldsymbol{R}$ or $\boldsymbol{D}_{\boldsymbol{m}}^{1/2}\boldsymbol{R}$, depending on the algorithm. This largest lambda can be efficiently calculated by doing a rank-restricted SVD. Then, the regularization path of length Λ, follows an exponential path starting at this value, going towards 1. Armed with this sequence of λ's we can construct the outline of the procedure for our three algorithms.

Outline 1: Algorithm outline for `WSI`, `RWSI` and `HRWSI` algorithms

input : Matrices $\boldsymbol{X}$ and $\boldsymbol{W}$ and sequence of λ's
output: Estimates $\widehat{\boldsymbol{AB}}'_{\lambda_1}, \widehat{\boldsymbol{AB}}'_{\lambda_2}, \ldots, \widehat{\boldsymbol{AB}}'_{\lambda_K}$

1 **begin**
2 $\boldsymbol{A}^{(n)}\boldsymbol{B}^{(n)\prime} \longleftarrow \boldsymbol{0}$;
3 **for** $\lambda_1 > \lambda_2 > \ldots > \lambda_K$ **do**
4 **while** $\frac{L_q(\boldsymbol{A}^{(o)},\boldsymbol{B}^{(o)}) - L_q(\boldsymbol{A}^{(n)},\boldsymbol{B}^{(n)})}{L_q(\boldsymbol{A}^{(n)},\boldsymbol{B}^{(n)})} < \epsilon$, *with* $q \in \{1,2,3\}$. **do**
5 set $\boldsymbol{A}^{(o)}\boldsymbol{B}^{(o)\prime} = \boldsymbol{A}^{(n)}\boldsymbol{B}^{(n)\prime}$;
6 Compute (18), (31) or (37) depending on the algorithm. ;
7 Calculate the updates $\boldsymbol{A}^{(n)}\boldsymbol{B}^{(n)\prime}$ according to (17) or (30). ;
8 $\widehat{\boldsymbol{AB}}'_{\lambda_k} \longleftarrow \boldsymbol{A}^{(n)}\boldsymbol{B}^{(n)\prime}$

In Outline 1, the procedure for all algorithms is displayed. In line 4, we see that the stopping criterion is different for each algorithm and that it is based on the change in value of the loss function. This implies that for steeper objective functions, the stopping criterion could be triggered later than for flatter objective functions. The algorithms only differ in line 4, 6, 7, and in their sequence of λ's. The sequence of estimates that is returned can be used for evaluation. Eventually, we will select a λ and corresponding rank for which the estimate has the best out-of-sample performance.

3.7 Computational Complexity

In order to gain some understanding on the expected performance in terms of speed, we investigate the computational complexity of the three algorithms. Let $|\Omega|$ be the amount of elements in the sparse part of our sparse-plus-low-rank matrices. We will construct and compare the complexity of all three algorithms. Recall that $\boldsymbol{A}$ is a $n \times k$ and $\boldsymbol{B}$ is an $p \times k$ matrix.

In line 4, we need to evaluate different objective functions for each iteration. A quick evaluation off (9) and (20) shows us that the `WSI` and `RWSI` have the same computational costs. For the `HRWSI` algorithm, we need to evaluate all $1 - \boldsymbol{a}_i'\boldsymbol{b}_j$

that fall in the sparse part, which costs an additional $O(|\Omega|)$, while the rest of the objective function is the same in terms of computational costs.

In line 6, the matrix to be decomposed needs to be constructed. In (18), matrix $((m-1)/m)\boldsymbol{A}^{(o)}$ needs to be explicitly formed, which requires $O(nk)$ flops since $(m-1)/m$ is already known. In addition, the sparse part of the matrix needs to be explicitly formed. Taking into account the matrices that do not change over iterations, this requires $O(2|\Omega|+|\Omega|(2k-1))$ flops. The costs are therefore $O(2|\Omega|+|\Omega|(2k-1)+nk)$ flops per iteration. In (31), this is exactly the same because most matrices are also known beforehand. in (37), we have an additional β_{ij}. Hence, we need to evaluate all $\boldsymbol{a}_i'\boldsymbol{b}_j$, which costs $O(|\Omega|)$. If they are larger than one we need to add $(w_{ij}/m_i)\beta_{ij}$ to the spare part. This will cost us $O(2|\Omega|)$ at most. Hence, the worst case complexity is $O(5|\Omega|+|\Omega|(2k-1))$.

In line 7, we conduct a SVD with the soft threshold SVD from Hastie et al. (2015), which takes $O(k|\Omega|+(n+p)k^2)$ flops and is equal for all algorithms. The only difference here is that the row-weighted algorithms needs to be pre-multiplied with a diagonal matrix, which takes another $O(nk)$ flops.

Hence, the total computational costs for each of the the three algorithms is:

- `WSI`: $O(2|\Omega|+|\Omega|(2k-1)+k|\Omega|+(n+p)k^2+nk)$
- `RWSI`: $O(2|\Omega|+|\Omega|(2k-1)+k|\Omega|+(n+p)k^2+2nk)$
- `HRWSI`: $O(6|\Omega|+|\Omega|(2k-1)+k|\Omega|+(n+p)k^2+2nk)$

Since they all differ only in the constants they are asymptotically equally complex. Nonetheless, we expect the runtimes per iteration to be of increasing order. This does not say much about the total runtime of total algorithm. Namely, we already established in Section 3.3 and 3.4 that the `WSI` algorithm's update might change slowly in the presence of large weights. Note that all algorithms scale linearly in complexity with the size of the data.

3.8 Evaluation Metric

The out-of-sample performance should always be assessed by means of some evaluation metric. Because implicit feedback consists only of a possible positive feedback, we cannot use any precision-oriented evaluation measures that depend on the size of the (predicted) ratings such as the Root Mean Squared Error or the Mean Absolute Error. For Implicit data, the most used evaluation metric is the recall-oriented Mean Percentage Ranking (MPR) value (Hu et al., 2008; Johnson, 2014).

We calculate the MPR from the predicted preference estimate $\widehat{\boldsymbol{AB}}'$. For each user, we generate a ranked list of the size of his predicted preferences. From this ranked list, we construct the percentile ranking: $rank_{ij}$. To clarify, for each user i

we set $rank_{ij} = 0\%$ for the most preferred item. For items in that are less preferred, the percentile ranking increases linearly with steps of $\frac{100\%}{p}$ where p is the amount of columns of $\boldsymbol{X}$, and thus the total amount of items. Percentile ranks are used rather than absolute ranks in order to make the value of the metric independent of p. The MPR score is defined as the weighted average of these percentile rankings for observations in the test set:

$$MPR = \frac{\sum_{ij} w_{ij}^t rank_{ij}}{\sum_{ij} w_{ij}^t}, \tag{38}$$

where the w_{ij}^t denote the weights that correspond to the test set. The expected MPR score for random prediction is 50% and lower values are more desirable. Lower values indicate that the observations in the test set have higher weighted average predicted preference.

Two drawbacks of the MPR score are that the optimal value depends on the size of the test set and that it depends on the selected weighting scheme. Firstly, we illustrate how it depends on the size of the test set. Take, for example, the case where for each user 10 observations are in the test set versus the case where 5 observations are in the test set. Then, if there are 100 items in total and these observations have equal weights. The optimal value for that row will be 5% in the first and 2.5% in the second case. As a result, it is hard to compare MPR scores across different datasets. Secondly, the MPR score is directly influenced by the selected weighting scheme through weights w_{ij}^t. As a consequence, it is hard to compare the performance of different weighting schemes using this metric.

In this research however, the MPR scores will only be used to assess the performance of the algorithms on the same dataset, with the same weighting scheme. Therefore, the MPR metric can be properly applied in our context.

4 Simulation Study

4.1 Simulation Setup

In this simulation study, we investigate the performance of our three algorithms in terms of speed and their ability to uncover the true underlying preferences. Recommender systems often need to be applied in situations with widely varying sparse matrices $\boldsymbol{X}$. It is therefore important to know the influence of the sparseness and matrix size on the performance of the algorithm.

We simulate data according to a relatively simple process. We generate the underlying preferences by drawing two standard normal random matrices $\widetilde{\boldsymbol{A}}$ and $\widetilde{\boldsymbol{B}}$ with dimensions $n \times k$ and $p \times k$ respectively. The underlying preferences are given by $\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{B}}'$ and we construct our artificial feedback matrix $\widetilde{\boldsymbol{X}}$ by $\tilde{x}_{ij} = 1 \iff \tilde{\boldsymbol{a}}_i'\tilde{\boldsymbol{b}}_j > 0$ and zero's elsewhere. Implicitly, we assume here that the user has positive preference for half of the products. The corresponding matrix $\widetilde{\boldsymbol{\Psi}}$ is generated according to $\tilde{\psi}_{ij} = \lceil \tilde{\boldsymbol{a}}_i'\tilde{\boldsymbol{b}}_j \rceil$, with $\lceil \cdot \rceil$ denoting the ceiling function. From $\widetilde{\boldsymbol{X}}$ and $\widetilde{\boldsymbol{\Psi}}$, we generate a test and a training set by sampling $(\delta \times 2)$ of the observed values in each row for the training set and leaving the rest in the test set. Because we have that $\tilde{x}_{ij} = 1 \iff \tilde{\boldsymbol{a}}_i'\tilde{\boldsymbol{b}}_j > 0$, δ corresponds to the sparseness level. Matrix $\widetilde{\boldsymbol{\Psi}}$ is split up accordingly. Note that in this setup our expected optimal MPR score lies around $\frac{50\%-\delta}{2}$, depending on the weights.

In order to have a good variety of matrices with different structures and levels of sparseness, we vary: $n = (1000, 1500, 15000)$, $p = (300, 750)$, $k = (10, 20)$, $\delta = (5\%, 2.5\%, 1.5\%)$. We run our algorithms again on these 36 combinations with 10% of the rows in $\widetilde{\boldsymbol{\Psi}}$ inflated by a factor 5, so we can investigate the effect of extreme weights on the `WSI` and `RWSI` algorithms. In total we have 72 datasets on which we run the three algorithms according to the procedure from Section 3.6, with a regularization path of 30 λ's ($\Lambda = 30$). We restrict the matrix rank of the estimate with a maximum of 30 to avoid costly computations. For each run of an algorithm, we save the lowest obtained MPR score with its associated matrix rank and value of the objective function. In addition, the total time for the algorithm to run over its regularization path is recorded in seconds.

The matrix rank of the estimate that is associated with the lowest MPR score is expected to be equal to k. That is, we expect the algorithm to achieve the lowest out-of-sample MPR score for estimates that have the same rank as our simulated matrices. Estimates with lower ranks are not likely able to correctly estimate $\widetilde{\boldsymbol{X}}$ because they cannot capture the more complex structure, while higher ranks are associated with overfitting. We investigate the ability of the algorithm to uncover the true underlying preferences through the difference between the uncovered rank and true rank k. This is more appropriate than MPR scores since these cannot be compared across datasets, as was discussed in Section 3.8

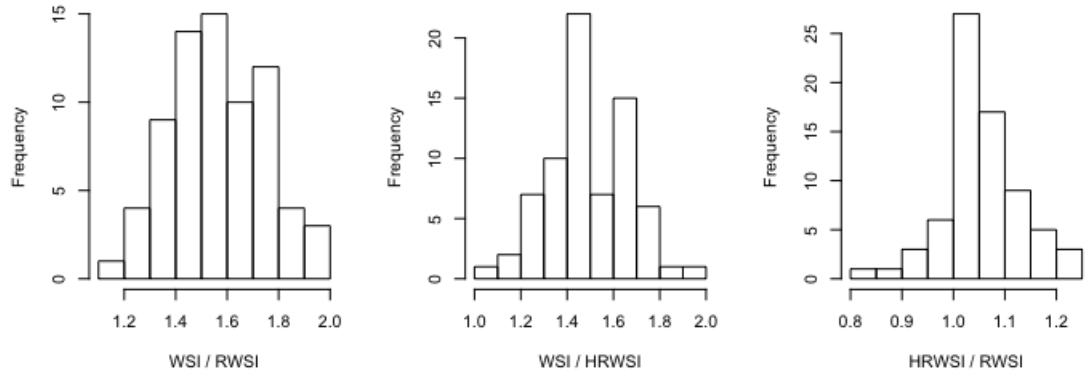


Figure 3: Histograms of the ratios between the total runtime of the algorithms

4.2 Results Simulation Study

In order to assess the performance of the three algorithms in terms of speed, we calculate the ratios of total running times over the entire regularization path. From the histograms in Figure 3, one can see that the `HRWSI` and `RWSI` have very comparable running times. Nonetheless, the `HRWSI` is often slightly slower than the `RWSI` algorithm. This likely due to the difference in computational complexity, while the variation is caused by different (local) optima. In addition, we find that the `WSI` algorithm is always slower than both the `HRWSI` as well as the `RWSI` algorithm, even though its iterations are computationally less intense. This result supports the idea that the `WSI` algorithm does not converge as fast as the row weighted algorithms because it uses the overall maximum in the majorization step (see Section 3.3). We expect this effect to increase in the presence of extreme weights for the `WSI` algorithm, while the other algorithms are more robust to this variation. Specifically, the `HRWSI` and `RWSI` algorithm are expected to have more similar running times with our without extreme weights, since the row-maxima are used in the majorization step. In Figure 4, we find support for this claim. The `HRWSI` and `RWSI` algorithms tend to be closer to the diagonal than the `WSI` algorithm. The effect becomes more visible for larger matrices with longer running times.

We proceed to investigate the algorithm's ability to uncover the true underlying preferences by investigating the matrix ranks of the estimates that obtain the lowest MPR. In Figure 5, we plotted the difference in the rank of the estimate and true underlying rank k against the lowest MPR scores that the algorithm obtained. From this plot, we identify two possible issues. First, there are many observations with MPR scores close to 50%. Apparently, the algorithms are unable to uncover the underlying structure and thus unable to improve over random prediction in these

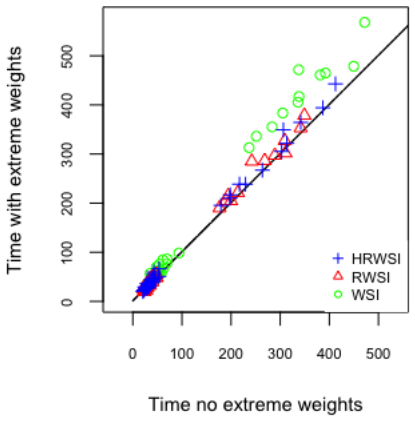


Figure 4: Running times (in seconds) in the presence of extreme weights plotted against the running times without extreme weights. The black line is the diagonal

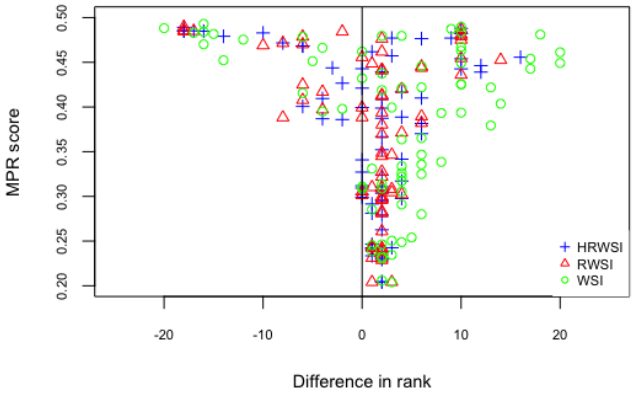


Figure 5: Differences between the uncovered rank (matrix rank of estimate with lowest out-of-sample MPR) and the true rank k plotted against their associated out-of-sample MPR score

cases. Another observation is that the `WSI` algorithm appears to overestimate k in a larger extent than the other algorithms. Both issues will be addressed sequentially.

In order to gain understanding of the first problem, we investigate the group of 39 observations with a MPR larger than 47.5%. This group consists of runs of the algorithm on 13 distinct datasets. All datasets have an amount of rows, n, of 1000 or 1500, they are all sparser than 5% and 11 of them have large underlying matrix ranks ($k = 20$). An explanation for the inability of the algorithms to uncover the true rank and obtain a lower MPR score is found in the binary nature of $\widetilde{\boldsymbol{X}}$. When we transform $\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{B}}'$ into $\widetilde{\boldsymbol{X}}$ we discard a lot of information, including information on the underlying rank. When matrices are small, binary and sparse, there is little information available and the information that is available is distorted. This makes it difficult for the algorithms to recover the underlying structure. In Figure 6, we find support for this claim. We see that the sparseness level has the largest effect on the obtained MPR score, followed by the true rank. Figure 6 suggests that this problem decreases with matrix size, even if the matrix is very sparse. Since recommender systems primarily deal with very large matrices, we believe that this problem will become nearly irrelevant in practice.

The second observation from Figure 5 was that the optimal estimates of the `WSI` algorithm often have higher matrix ranks than the optimal estimates of the other algorithms. Closer inspection shows us that for these observations, the algorithm ran on datasets with inflated weights for 10% of the rows. In Figure 7 and 8, we see that

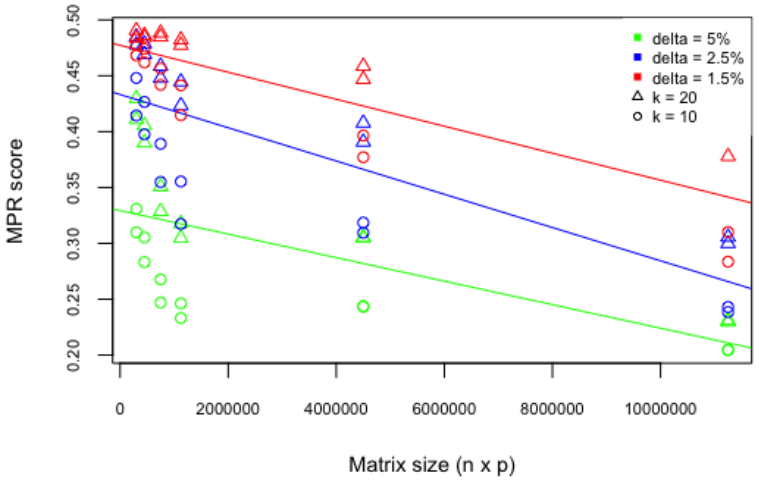


Figure 6: Average MPR scores for different matrix sizes ($n \times p$), with varying sparseness levels (δ) and true ranks (k). Trendlines are constructed for each sparseness level.

under similar conditions (sparseness: $\delta = 5\%$ and true rank: $k = 10$), the uncovered ranks tend to increase for the `WSI` algorithm in the presence of inflated weights. The `HRWSI` and `RWSI` algorithms are more robust to extreme weights and do not change much. A likely explanation is that the inflated weights hinder the convergence of the algorithm. From Section 3.3, we know that extreme weights cause the new estimate to be fitted mostly to old estimate. This potentially leads to premature triggering of the stopping criterion. Then, for one of the next lambdas in the regularization path, an estimate with higher rank might achieve a lower MPR score and will be selected as the optimal estimate. This effect is reduced in the `HRWSI` and `RWSI` algorithms by using row maximum weights. For the `WSI` algorithm, it is important to reduce the effect of extreme weights by adopting an appropriate weighting scheme.

In this simulation analysis we have found that our algorithms potentially run into problems if they need to estimate sparse matrices. Fortunately, we found evidence that the sparsity problem reduces for larger matrices, which are typically used in the recommender system context. In addition, we found that the `WSI` algorithm performs poorly in the presence of extreme weights, both in terms of speed as in its ability to uncover the true rank of the estimate. If we keeps these two points in mind, we believe that the algorithms will show good performance in practice.

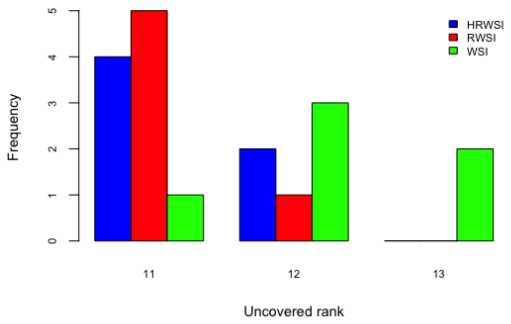


Figure 7: The uncovered ranks for each algorithm without inflated weights.

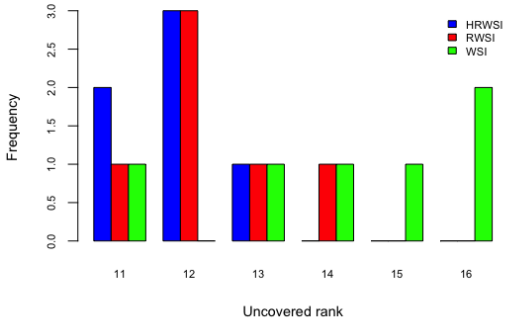


Figure 8: The uncovered ranks for each algorithm with inflated weights.

5 Empirical example

We use an empirical dataset from Helloprint, an e-commerce company that specializes in print, to illustrate the practical use of our algorithms. They offer a wide variety of products, ranging from booklets to printed kitchen knifes. The data used consists of online clickdata on products. In this analysis, we only include users that have provided feedback for more than one product because the company focuses on returning customers. We have in total 571,179 observed clicks from 159,279 users on 258 products. These products are defined with the objective of our recommender system in mind. That is, we want to recommend products that the user is unaware of and that he likes. We treat for example "flyers A4" and "flyers A5" as one product, since a user will be aware that both are offered after having ordered one.

The purpose of this empirical example is to show that the algorithm works in practice. We acknowledge that the choice of the weighing scheme and the tuning of the parameters, has a large impact on the results. Because computations are costly and this article focuses on the development of the algorithms, we do not evaluate the performance of all possible weighting schemes in this empirical example. As we want to avoid the influence of extreme weights, we choose the logarithmic weighting scheme. We set $\alpha \in \{7.5, 15, 22.5, 30\}$ and look for the value for which we obtain the lowest out-of-sample MPR.

For 10% of all users, we randomly take out 1 observation as a test set. This approach is adopted to prevent a situation where all observations of a user are drawn into the test set, thereby deleting the user's information from the training data. As a result, we have 555,252 observations in our training- and 15,927 in our test-set.

In order to better assess the performance of the algorithms, we also report the

MPR score of a competing model. We use the popularity model, which simply recommends the most popular products to all users. This model is surprisingly powerful since users often concentrate on relatively few products.

For each of the algorithms, we execute the estimation procedure from Section 3.6 with a regularization path of 50 lambdas. Since the regularization paths are different, we will compare the out-of-sample MPR scores for different ranks of the estimates $\widehat{\boldsymbol{AB}}'_{\lambda_i}$. In order to improve the ease of comparison, we restrict the rank of each estimate to be at most 2 higher than the previous, with a maximum of 30 to save computational time. By doing so, we create a grid of ranks with corresponding MPR scores that can easily be compared.

For this dataset, we found that the lowest MPR was reached for $\alpha = 30$. In Figure 9, we display the MPR scores obtained by each algorithm for $\alpha = 30$.

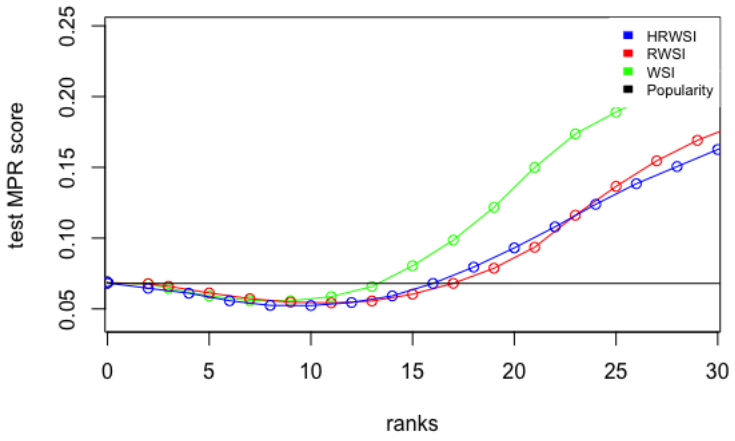


Figure 9: MPR scores with the associated rank of the estimate $\widehat{\boldsymbol{AB}}'_{\lambda_i}$ for $\alpha = 30$

As a first observation in Figure 9, we would like to note that the Popularity method shows an extremely good performance in terms of MPR. This is explained by the fact that there are only a few products very popular at Helloprint, while most other products are only visited occasionally. As a result, the MPR score of 0.068 for the popularity method is already a large improvement to the expected MPR for random prediction of 0.5.

Nonetheless, we see that all algorithms are able to outperform the popularity method in terms of MPR. The best out-of-sample performance is attained around rank 8 - 12 for all algorithms. As a minimum MPR score, we find for `HRWSI` 5.23%, for `RWSI` 5.43% and for `WSI` 5.56%, with associated values of λ of 185.73, 185.73 and 1743.40 respectively. It would be wrong to conclude that the `HRWSI` algorithm shows

the best performance since these minima are not necessarily the true minimal values. As we search only over 50 lambdas, we could improve our prediction by searching for a better lambda in the neighborhood of our optimal values with more flexibility on the ranks of the estimate. Nonetheless, we may conclude that our algorithms show superior performance in comparison to the the popularity method.

On this dataset, the `RWSI` algorithm structurally had the shortest computation time. For $\alpha = 30$, this was 34 minutes. The `HRWSI` and `WSI` algorithms took 41 and 46 minutes to compute. Calculations were done on an Amazon EC2 instance of type "`c4.4xlarge`", with 8 processor cores and 15GB of RAM. The runtime for lower levels of α were shorter since an increase in α leads to more extreme weights, which slows down the algorithm.

6 Conclusion and Discussion

In this paper, we aimed to extend the convex spectral regularization methods for matrix factorization to the case of implicit data. Until now, all available methods used Gradient Descent or ALS type techniques for optimization. Because convex minimization is quick and does not need a lot of observed data points, it seems to be an attractive alternative over the current methods.

The three algorithms that were developed seem to show good performance. In our simulation studies we saw that the algorithms are, under certain conditions, able to uncover the underlying structure. We uncovered two possible problems that should be taken into account when the algorithms are applied in practice. Firstly, we have seen that the methods fail on small and sparse matrices, but that this effect disappears for large matrices. Because the typical user-item matrix is very sparse, we should assure us that they are sufficiently large. Secondly, in the presence of extreme weights, the `WSI` algorithm becomes unreliable. This can be solved by adopting a different weighting scheme.

We conclude that from our three algorithms, the `HRWSI` and `RWSI` algorithms outperform the `WSI` algorithm in terms of speed and robustness to extreme weights. We cannot base any conclusions on the predictive performance of our algorithms, since it could depend on the use case and more tuning of the parameters is required. Nonetheless, we have seen that all our methods outperform the popularity recommendation method.

For future research, it is of interest to test the performance of the algorithms against some ALS or Gradient descent type methods such as the ones discussed in the related works section. It should be verified if the proposed methods show comparable, or even better, performance on a variety of datasets. Moreover, the algorithms should be run on an implicit feedback dataset that is widely used in academic research. In this way, the performance in terms of speed can be easily compared with other algorithms. Unfortunately, there is not (yet) such a dataset openly available.

Future research on the performance of the one-sided quadratic hinge loss model for implicit data is needed as well. The method can easily be extended to a two-side variant where we include implicit positive as well as negative implicit feedback. Such negative implicit feedback could consist of the amount of complaints or returning products. It is of interest to see whether this improves the performance of the method.

In addition, more research is needed on the effectiveness of different weighting schemes since a clear framework is currently missing. Depending on the situation, it could be that temporal or contextual information should be included while for other situations this does not make any sense.

Finally, it is of interest to continue developing new implicit-feedback algorithms

that use convex optimization techniques. To the best of our knowledge, this article is the first that uses convex optimization techniques for implicit-feedback spectral regularized algorithms. Many alternative algorithms can be developed that exploit the problem structure in a similar way as was done in this article. Exploiting these new algorithms could lead to even more accurate recommendations and a further reduction of the choice overload problem.

References

Angst, R., Zach, C., and Pollefeys, M. (2011). The generalized trace-norm and its application to structure-from-motion problems. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2502–2509. IEEE.

Bennett, J., Lanning, S., et al. (2007). The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, NY, USA.

Candes, E. J. and Recht, B. (2008). Exact low-rank matrix completion via convex optimization. In *Communication, Control, and Computing, 2008 46th Annual Allerton Conference on*, pages 806–812. IEEE.

Fang, Y. and Si, L. (2011). Matrix co-factorization for recommendation with rich side information and implicit feedback. In *Proceedings of the 2nd International Workshop on Information Heterogeneity and Fusion in Recommender Systems*, pages 65–69. ACM.

Fazel, M. (2002). *Matrix rank minimization with applications*. PhD thesis, PhD thesis, Stanford University.

Greenacre, M. J. (1984). Theory and applications of correspondence analysis.

Groenen, P., Giaquinto, P., and Kiers, H. (2003). Weighted majorization algorithms for weighted least squares decomposition models. Technical report.

Groenen, P. J., Nalbantov, G., and Bioch, J. C. (2008). Svm-maj: a majorization approach to linear support vector machines with different hinge errors. *Advances in data analysis and classification*, 2(1):17–43.

Hastie, T., Mazumder, R., Lee, J. D., and Zadeh, R. (2015). Matrix completion and low-rank svd via fast alternating least squares. *J. Mach. Learn. Res*, 16(1):3367–3402.

Hu, Y., Koren, Y., and Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 263–272. Ieee.

Jain, P., Netrapalli, P., and Sanghavi, S. (2013). Low-rank matrix completion using alternating minimization. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 665–674. ACM.

Johnson, C. C. (2014). Logistic matrix factorization for implicit feedback data. *Advances in Neural Information Processing Systems*, 27.

Koren, Y. (2008). Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434. ACM.

Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, 42(8).

Kristof, W. (1970). A theorem on the trace of certain matrix products and some applications. *ETS Research Report Series*, 1970(1).

Lee, T. Q., Park, Y., and Park, Y.-T. (2008). A time-based approach to effective recommender systems using implicit feedback. *Expert systems with applications*, 34(4):3055–3062.

Mazumder, R., Hastie, T., and Tibshirani, R. (2010). Spectral regularization algorithms for learning large incomplete matrices. *Journal of machine learning research*, 11(Aug):2287–2322.

Oard, D. W., Kim, J., et al. (1998). Implicit feedback for recommender systems. In *Proceedings of the AAAI workshop on recommender systems*, pages 81–83. Menlo Park, CA: AAAI Press.

Ortega, J. M. and Rheinboldt, W. C. (1970). *Iterative solution of nonlinear equations in several variables*. Academic Press, New York.

Pan, R., Zhou, Y., Cao, B., Liu, N. N., Lukose, R., Scholz, M., and Yang, Q. (2008). One-class collaborative filtering. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 502–511. IEEE.

Pilászy, I., Zibriczky, D., and Tikk, D. (2010). Fast als-based matrix factorization for explicit and implicit feedback datasets. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 71–78. ACM.

Takane, Y. and Shibayama, T. (1991). Principal component analysis with external information on both subjects and variables. *Psychometrika*, 56(1):97–120.

A Appendix

A.1 Definition of WNN

We use the approach from Angst et al. (2011) to arrive at the definition of the Weighted Nuclear Norm (WNN). A summary of their approach is given in this appendix.

We know that the Nuclear Norm (NN) is defined as the sum of singular values, hence:

$$||\boldsymbol{X}||_* = \sum_{i}^{\min(m,p)} \sigma_i. \tag{39}$$

There exists, however, other equivalent ways to define the NN. One way is to write it as a Semi Definite Programming problem:

$$\begin{aligned} ||\boldsymbol{X}||_* = \frac{1}{2} \min_{\boldsymbol{X}_1 \boldsymbol{X}_2} &\langle \boldsymbol{I}_{m+n}, \begin{bmatrix} \boldsymbol{X}_1 & \boldsymbol{X} \\ \boldsymbol{X}' & \boldsymbol{X}_2 \end{bmatrix} \rangle \\ \text{s.t.} \quad &\begin{bmatrix} \boldsymbol{X}_1 & \boldsymbol{X} \\ \boldsymbol{X}' & \boldsymbol{X}_2 \end{bmatrix} \succeq \boldsymbol{0}_{(m+n)\times(m+n)}. \end{aligned} \tag{40}$$

This definition allows us to arrive at the WNN. If we take positive definite block diagonal matrix and it's eigen-decomposition:

$$\begin{aligned} \boldsymbol{D} &= \begin{bmatrix} \boldsymbol{D}_r & \boldsymbol{0} \\ \boldsymbol{0}' & \boldsymbol{D}_c \end{bmatrix} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}' \\ \boldsymbol{V} &= \begin{bmatrix} \boldsymbol{V}_r & \boldsymbol{0} \\ \boldsymbol{0}' & \boldsymbol{V}_c \end{bmatrix}, \qquad \boldsymbol{\Lambda} = \begin{bmatrix} \boldsymbol{\Lambda}_r & \boldsymbol{0} \\ \boldsymbol{0}' & \boldsymbol{\Lambda}_c \end{bmatrix}, \end{aligned}$$

we can replace the identity matrix with $\boldsymbol{D}$. Then, using the cyclical property of the trace function, we have that:

$$\begin{aligned} ||\boldsymbol{X}||_{*\boldsymbol{D}} &= \frac{1}{2} \min_{\boldsymbol{X}_1 \boldsymbol{X}_2} \langle \boldsymbol{D}, \begin{bmatrix} \boldsymbol{X}_1 & \boldsymbol{X} \\ \boldsymbol{X}' & \boldsymbol{X}_2 \end{bmatrix} \rangle \\ &= \frac{1}{2} \min_{\boldsymbol{X}_1 \boldsymbol{X}_2} \text{trace} \left(\boldsymbol{D}' \begin{bmatrix} \boldsymbol{X}_1 & \boldsymbol{X} \\ \boldsymbol{X}' & \boldsymbol{X}_2 \end{bmatrix} \right) \\ &= \frac{1}{2} \min_{\boldsymbol{X}_1 \boldsymbol{X}_2} \text{trace} \left(\boldsymbol{\Lambda}^{1/2} \boldsymbol{V}' \begin{bmatrix} \boldsymbol{X}_1 & \boldsymbol{X} \\ \boldsymbol{X}' & \boldsymbol{X}_2 \end{bmatrix} \boldsymbol{V} \boldsymbol{\Lambda}^{1/2} \right) \\ &= \frac{1}{2} \min_{\boldsymbol{X}_1 \boldsymbol{X}_2} \text{trace} \left(\begin{bmatrix} \boldsymbol{C}_r \boldsymbol{X}_1 \boldsymbol{C}_r' & \boldsymbol{C}_r \boldsymbol{X} \boldsymbol{C}_c' \\ \boldsymbol{C}_c \boldsymbol{X}' \boldsymbol{C}_r' & \boldsymbol{C}_c \boldsymbol{X}_2 \boldsymbol{C}_c' \end{bmatrix} \right), \end{aligned} \tag{41}$$

where $\boldsymbol{C}_r = \boldsymbol{\Lambda}_r^{1/2} \boldsymbol{V}_r'$ and $\boldsymbol{C}_c = \boldsymbol{\Lambda}_c^{1/2} \boldsymbol{V}_c'$. If we now write $\boldsymbol{Y}_1 = \boldsymbol{C}_r \boldsymbol{X}_1 \boldsymbol{C}_r'$ and $\boldsymbol{Y}_2 = \boldsymbol{C}_c \boldsymbol{X}_2 \boldsymbol{C}_c'$, we conclude from (41) that we have: $||\boldsymbol{C}_r' \boldsymbol{X} \boldsymbol{C}_c||_* = ||\boldsymbol{X}||_{*\boldsymbol{D}}$