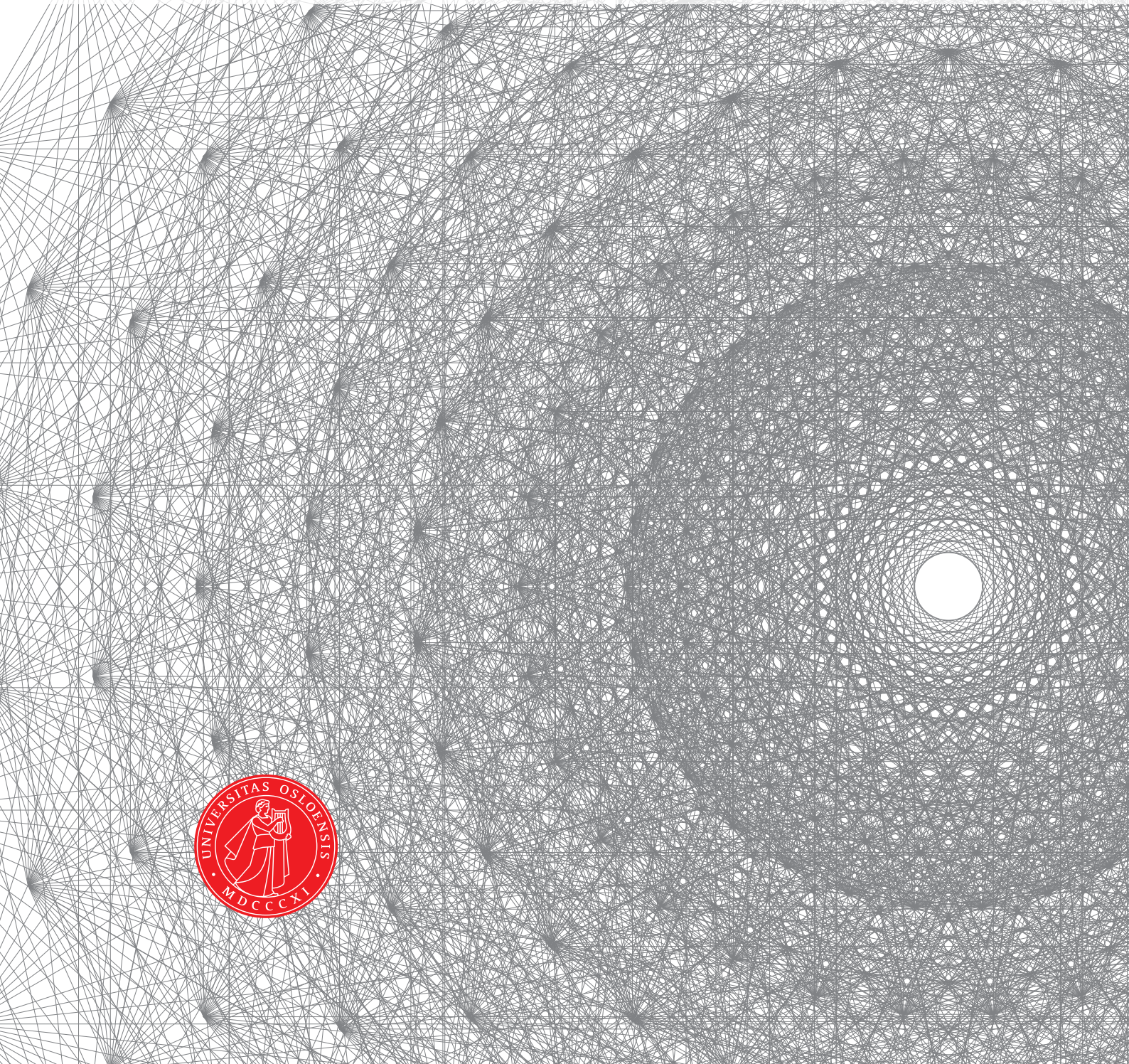
UiO : Department of Mathematics
University of Oslo

Collaborative Filtering for Implicit Feedback

Investigating how to improve NRK TV's recommender system by including context

Jonas F. Schenkel

Master's Thesis, Spring 2017



This master's thesis is submitted under the master's programme *Modelling and Data Analysis*, with programme option *Statistics and Data Analysis*, at the Department of Mathematics, University of Oslo. The scope of the thesis is 60 credits.

The front page depicts a section of the root system of the exceptional Lie group E_8, projected into the plane. Lie groups were invented by the Norwegian mathematician Sophus Lie (1842–1899) to express symmetries in differential equations and today they play a central role in various parts of mathematics.

Abstract

Collaborative filtering for implicit feedback has become an important tool for helping users navigate the web. One of the services that utilize its power is NRK TV who base their recommendations on the weighted regularized matrix factorization (WRMF) model. This thesis reviews two traditional latent factor models with matrix factorization, one of them being the WRMF and the other is based on a Bayesian criterion. Both models are then applied to data from NRK TV, where WRMF performs better.

NRK TV has much more information available than what is used in WRMF, and the main goal of this thesis has been to improve recommendations by utilizing more of the available information. The author shows how we can use higher order arrays to model contextual information via tensor decomposition. Collaborative filtering with tensor decomposition is considered to be state of the art, and the author shows how one can use CP and Tucker decomposition to give recommendations that are tailored to the situation. The model involving Tucker decomposition is, to the authors knowledge, not detailed in the literature. The applications of these methods on the NRK TV data were however, limited.

By including information about time of event, the author proposes a somewhat improved version of WRMF that is easily implemented. This modification allows for users taste to drift with time. It can for example be used to recommend users newer content that fit their taste profiles, or, with small modifications, give users seasonal recommendations, for example for Christmas time.

Acknowledgements

First and foremost, I am very thankful to my team of supervisors: Linn Cecilie Solbergersen, Arnoldo Frigessi, Ørnulf Borgan and Ida Scheel. Linn Cecilie for her inside knowledge on recommender systems and NRK. Arnoldo for his new perspectives and bigger questions. Ørnulf for not allowing me to start on more than I could finish and his «nitpicking». And especially to my main supervisor Ida, for weekly talks, discussions and feedback throughout the process. I was lucky to get quality as well as quantity in my supervisors. They have, no doubt, increased the quality of this thesis.

This thesis was produced in cooperation with Big Insight. Being part of this project was an educating experience. The talks hosted by Big Insight were welcome breaks from writing, and I'm grateful that I was treated like a member of the project.

NRK kindly gave me access to a very interesting dataset with implicit feedback. I am grateful for the kind welcome I received by Linn Cecilie's team, on my visits to NRK.

To my fellow students at Utfallsrommet for all the coffee, and for helping me make UiO my second home.

To my parents, Jan Hugo and Ingunn, for supporting me when Lånekassa would not.

And to Adriane, for energy.

Blindern, 2017
Jonas F. Schenkel

Contents

Chapter 1

Introduction

1.1 Recommender Systems

In the last decade the way we consume and buy content has changed. Internet services such as Netflix[1], Spotify[2] and NRK TV[3] have made entertainment available at our fingertips, at any time. Online retailers like Amazon[4] carry millions of products in different product categories. These sites have huge catalogs of items that the user can interact with or buy. An inconvenience with these content rich web services, is the time the user has to spend navigating the services, in order to find the content he wants to interact with. Many have spent hours of their youth at video rental shops trying to decide which movie to rent. These rental shops often had a selection of hundreds of movies to choose from. The catalogs at sites like Netflix are magnitudes greater. When the user is faced with millions of items it can be difficult to choose correctly, i.e. choose one of the more interesting items in the catalog. Schwartz (2004) introduced the paradox of choice, that faced with too many alternatives we end up less happy. We invest time in the choices we make and we have to invest more time if there are many alternatives. We also start second guessing our choices more when the number of alternatives increase (Schwartz 2004). Recommender systems (RS) try to help us choose when the alternatives are many, by accentuating the items that it believes we will find interesting. RS are information filtering systems that predict the user's preference for items and then recommend most relevant items. A good RS is good for customers, who find good content more easily, and for retailers who in return get higher sales.

The interest in the area of RS increased dramatically with the Netflix prize.[5] In October 2006 Netflix announced that the team that could beat their recom-

[1] `www.netflix.com`
[2] `www.spotify.com`
[3] `tv.nrk.no`
[4] `www.amazon.com`
[5] `www.netflixprize.com`

mendation algorithm with 10 % would receive $ 1,000,000. It was not until September 2009 that the 10 % threshold was broken. The interest in RS has not declined notably even after the prize was taken, mostly because of the growth of web services and online retailers. A lot of research goes into improving the user's experience by making the recommendations better. The most popular technique used by RS is what is known as collaborative filtering.

1.1.1 Collaborative Filtering

RS helps users find content they do not know they are looking for. The basic idea of collaborative filtering (CF) is that users who have enjoyed similar items in the past, also will do so in the future, hence the name collaborative filtering. CF methods only use the users' history. They do not take data about the entities, e.g. users and items, the meta data, into account. For users this can include for example information about age, sex and occupation. Meta data for items can include genre, length and producer or composer for the item. RS that uses such meta-data go under the label *content-based filtering*. If the RS combines collaborative and content-based filtering, we call it a hybrid RS. Because CF does not use meta-data, CF can perform well in domains where there is little to no information about the items. The flip side is that they suffer from what is called the *cold start* problem. Cold start is the problem that both items and users need to have a few past events to build up «knowledge» about them. If the user does not have enough events for the RS to learn his preferences, the recommendations he receives will not be well individualized. The same goes for items. Cold start is a big problem for dynamic content with a short relevant lifetime, such as news. Cold start will not be studied in this thesis, but is a popular topic in CF literature (Ahn 2008, Lam et al. 2008).

This thesis will look at CF methods applied to a dataset from NRK TV where events are composed of users watching programs. We will be interested in recommending programs that the user will find relevant. CF shines when personal taste is dominant such as with music and movies. It is able to give good personalized recommendations while content-based filtering is based on the idea of user stereotypes. CF is often divided into two subgroups: neighborhood models and latent factor models. Both will be reviewed in Chapter 3, but latent factor models will be the focus of this thesis. While the objective of RS is to give each user a best possible individualized top-N list of items, the user's preferences can be dependent on the situation.

1.1.2 Context Aware Recommender Systems

The situation in which the user and content interacts is in many cases not static. Movies and music are often enjoyed in the company of others, and the user might then prefer items that he would not consume alone. As an

example, the movies a user watches in the company of children will often differ from what the user watches in the company of friends. Data on contextual information like the companions, device and the mood of the user are often not available, but other data such as time and location are often registered. RS that use data about the situation in which the user and item interact are called *context aware* (Adomavicius & Tuzhilin 2015). Context aware RS adapt the recommendations to the given situation in which the content is consumed by the user to give the user more relevant recommendations. It can be to adjust which restaurants to recommend based on the weather, or recommending more upbeat songs in the morning. Context aware recommender systems can also find individualized differences across contexts. Traditional RS only consider the entities user and item as input when doing recommendations, they take the form $USER \times ITEM \rightarrow RECOMMENDATION$. Context aware includes a contextual factor such that it takes the form $USER \times ITEM \times CONTEXT \rightarrow RECOMMENDATION$.

Because much of what we choose is situation dependent, also online, including context to RS is seen as important. And although many of the factors that affect the choice are not observable, some of the important ones are. Time might be the most studied contextual factor, and we will see how it can be applied to the NRK data in Chapter 5.

1.2 Implicit vs Explicit Feedback

We leave different trails of data when we use the web. With CF it is important to distinguish between *explicit* and *implicit* feedback. With explicit feedback the user has expressed their opinion about the item. This is often done by rating. Netflix used a five-star system until recently, but has shifted to a thumbs-up and thumbs-down rating system. We have implicit feedback when we can only observe what the user does, i.e. which items he interacts with. These interactions can be listening to a song, watching half of a movie, putting an item into the shopping cart or going through with the purchase of said item. To not be intruding or bothersome, most sites only track the users movements and does not ask the user to rate items. Even when explicit feedback is registered, it usually only constitute a small part of the total data collected.

As mentioned, Netflix registers ratings and the dataset used in the Netflix prize had explicit feedback. The Netflix prize, and the fact that it is easier to work with, has lead the literature to mostly focus on explicit feedback. The focus of this thesis will be on implicit feedback where we only know if or how much the user has interacted with the item, not how much he appreciated it.

1.3 About Missing Data

One of the things about implicit feedback that makes it more interesting than explicit, is the missing data, i.e. the items the user has not interacted with. In the case of explicit data we know the rating the user gave the item, and if the user has not interacted with the item we assume that he does not know about it. For implicit data it is not that easy. It is natural to assume positive correlation between how much a user interacts with an item and how much the user enjoys the item. So an interaction between a user and an item is taken as a positive sign of preference, the user finds the item interesting. This leads us to only having positive observations. This is a problem, as positive does not say much when everything is positive. With implicit data a big problem is that we have no way of knowing which items the user dislikes. We have a problem of missing negatives. We solve this problem by making the assumption that not interacting with an item is a negative signal. One would not interact with an item that one does not find interesting. A single man might not find an online article about women's makeup interesting and he will not click on it. If he had to rate its relevancy, that is give explicit feedback, he would surely rate it as not interesting. But some of the items the user has not interacted with we believe he will actually enjoy. He just does not know it. This can be because he does not know the item exists or because he does not know that it is available on the service. These are the items we want to recommend users. How to treat the missing information is a very important aspect of doing CF with implicit feedback.

Chapter 2

Background Material

In this chapter I will present relevant, technical background material, that is needed in order to discuss the CF methods in chapter 3. The first section is about singular value decomposition, which is closely connected to the latent factor models, to be introduces later in Section 3.2. Notation and concepts for tensors, multi-dimensional arrays, is explained in Section 2.2, before the chapter closes with a quick review of the two optimization algorithms, that will be used to estimate the latent factor models, in Section 2.4.

2.1 Singular Value Decomposition

A factorization of a $U \times I$ matrix $\mathbf{A}$ with rank F, into the product of three matrices on the form

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T, \tag{2.1}$$

where $\mathbf{U}$ is $U \times U$ and $\mathbf{V}$ is $I \times I$, is always possible (Lay 2002). Moreover we have that both $\mathbf{U}$ and $\mathbf{V}$ are unitary, i.e. $\mathbf{U}\mathbf{U}^T = \mathbf{U}^T\mathbf{U} = \mathbf{I}$ and $\mathbf{V}\mathbf{V}^T = \mathbf{V}^T\mathbf{V} = \mathbf{I}$, and $\mathbf{S}$ is $U \times I$ where the first F diagonal entries are the F singular values of $\mathbf{A}$ in decreasing order, and the rest of the entries are zeros. A factorization on the form of (2.1) is called a singular value decomposition (SVD). The singular values of $\mathbf{A}$ are the square roots of the eigenvalues of the symmetric matrix $\mathbf{A}^T\mathbf{A}$. They are non-negative and ordered such that $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_F \geq 0$. SVD has been widely used in image processing and other fields where data compression is central, and it can be used to determine the rank of a matrix.

2.1.1 Best Rank *k*-approximation

The reason why SVD is so important in collaborative filtering with matrix factorization is through the truncated SVD. The truncated SVD gives a low rank approximation of the full matrix. For $k < F$ we can find a rank-k approximation of $\mathbf{A}$ in (2.1), by replacing the trailing singular values $\sigma_{k+1}, \sigma_{k+2}, \ldots, \sigma_F$ on the

diagonal of $\mathbf{S}$, with zeros. This gives us a new rank-k matrix $\mathbf{S}_k$ with diagonal entries $s_{ii} = \sigma_i$ for $i = 1, 2, \ldots, k$, and we get the rank-k approximation from the truncated SVD as $\mathbf{U}\mathbf{S}_k\mathbf{V}^T$. If the approximation is sufficiently good, instead of the original matrix, we only need to keep k columns of $\mathbf{U}$ and $\mathbf{V}$, along with the singular values. If U and I are big, this can be a big reduction in number of elements to store.

The Frobenius norm of a matrix $\mathbf{A} \in \mathbb{R}^{U \times I}$, with elements a_{ui}, is

$$\|\mathbf{A}\|_F = \sqrt{\sum_{u=1}^{U}\sum_{i=1}^{I} a_{ui}^2},$$

i.e. the square root of the sum of squared elements of the matrix. The truncated SVD gives the best low rank approximation of the original matrix when we use the Frobenius norm as criterion. Because the Frobenius norm will be used throughout the thesis and the proof of this property is short, it will be presented in the text. In the proof we will use that the Frobenius norm is unitarily invariant so given any unitary matrices $\mathbf{U} \in \mathbb{R}^{U \times U}$ and $\mathbf{V} \in \mathbb{R}^{I \times I}$, we have $\|\mathbf{U}^T\mathbf{A}\mathbf{V}\|_F = \|\mathbf{A}\|_F$.

Proof. We need to show that

$$\mathbf{U}\mathbf{S}_k\mathbf{V}^T = \arg\min_{\mathbf{B}} \|\mathbf{A} - \mathbf{B}\|_F \quad s.t. \quad \text{rank}(\mathbf{B}) \leq k,$$

for $\mathbf{A} \in \mathbb{R}^{U \times I}$ with rank $F \geq k$. We have

$$\begin{aligned}\|\mathbf{A} - \mathbf{B}_k\|_F &= \|\mathbf{U}\mathbf{S}\mathbf{V}^T - \mathbf{B}_k\|_F = \|\mathbf{S} - \mathbf{U}^T\mathbf{B}_k\mathbf{V}\|_F = \|\mathbf{S} - \mathbf{N}\|_F \\ &= \sqrt{\sum_{i=1}^{k}(\sigma_i - n_{ii})^2 + \sum_{i=k+1}^{F}(\sigma_i - n_{ii})^2 + \sum_{i \neq j} n_{ij}^2 + \sum_{i>r} n_{ii}^2}, \qquad (2.2)\end{aligned}$$

where we have used the invariance property in the second equality. We see that (2.2) is clearly minimized if $n_{ii} = \sigma_i$ for all $i \in \{1, 2, ..., k\}$ and all other entries are 0. □

This is an application of the famous Eckhart-Young theorem (Eckart & Young 1936) and shows an important property of SVD, the truncated SVD minimizes the sum of squared errors! This property is why many CF methods using matrix factorization have «SVD» in their name.

2.1.2 A Simple Example of Matrix Factorization

This section will show the principles of what is to come in the next chapter. We will be interested in approximating a given matrix $\mathbf{A} \in \mathbb{R}^{U \times I}$ by a factorization

into two low rank matrices $\mathbf{X}$ and $\mathbf{Z}$, such that $\mathbf{A} \approx \mathbf{X}\mathbf{Z}^T$. We can start with the SVD decomposition

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T = \mathbf{U}\mathbf{S}_1^{1/2}\mathbf{S}_2^{1/2}\mathbf{V}^T = \mathbf{X}\mathbf{Z}^T, \tag{2.3}$$

where $\mathbf{X} = \mathbf{U}\mathbf{S}_1^{1/2}$, $\mathbf{Z}^T = \mathbf{S}_2^{1/2}\mathbf{V}$, $\mathbf{S}_1^{1/2} \in \mathbb{R}^{U \times f}$ has elements $s_{1,ii}^{1/2} = \sqrt{\sigma_i}$ for $i = 1, 2, ..., f$ and all other elements zero; $\mathbf{S}_2^{1/2} \in \mathbb{R}^{f \times I}$ and constructed equivalently. We could have chosen to simply multiply the matrix $\mathbf{S}$ into one of the other matrices, but the way shown in (2.3) is best for illustrative purposes. Finding the matrices $\mathbf{X}$ and $\mathbf{Z}$ like this is at the core of matrix factorization techniques in collaborative filtering.

Table 2.1: Simple example of users' movie ratings

	The Matrix	Speed	Casablanca	Titanic	This Is Spinal Tap	Philadelphia
Chuck	5	5	3	1	1	5
Julia	1	2	5	4	1	4
Arnold	4	4	1	2	5	2
Hugh	1	1	5	4	4	2

Now that we have a way of finding what we will come to know as the user feature matrix $\mathbf{X}$ and item feature matrix $\mathbf{Z}$, we can apply this to some made up data. If we consider the matrix of users' movie ratings in Table 2.1, we see that we have four users who all have rated six movies on a scale from 1 to 5. A common problem with ratings is that users rate movies differently. Some tend to rate all items higher and some are very critical. Therefore, we will work with de-meaned rows, i.e. we subtract the row's mean from each row. So we have

$$\mathbf{A} = \begin{pmatrix} 1.7 & 1.7 & -0.3 & -2.3 & -2.3 & 1.7 \\ -1.8 & -0.8 & 2.2 & 1.2 & -1.8 & 1.2 \\ 1.0 & 1.0 & -2.0 & -1.0 & 2.0 & -1.0 \\ -1.8 & -1.8 & 2.2 & 1.2 & 1.2 & -0.8 \end{pmatrix}.$$

This makes it easier to see how a user has rated the movies differently. If we do SVD on the matrix with de-meaned rows $\mathbf{A}$ we end up with a matrix $\mathbf{U} \in \mathbb{R}^{4\times4}$, the matrix $\mathbf{S} \in \mathbb{R}^{4\times6}$ consisting of the singular values and the matrix $\mathbf{V} \in \mathbb{R}^{6\times6}$. If we create $\mathbf{S}_1^{1/2}$ and $\mathbf{S}_2^{1/2}$ and do the multiplication in the same way as above we end up with the two matrices $\mathbf{X} \in \mathbb{R}^{4\times4}, \mathbf{Z} \in \mathbb{R}^{6\times4}$. Now we have actually mapped the users and items to the same latent space. This is the very idea of CF with matrix factorization. We can look at how the coordinates of the first two latent variables look for the users and items in Figure 2.1. We can see that Julia and Hugh have similar feature combinations as Casablanca and Titanic, both very romantic movies that they have given high ratings. Arnold and Chuck are on the other side of feature 1 with the more action packed movies. If we use only

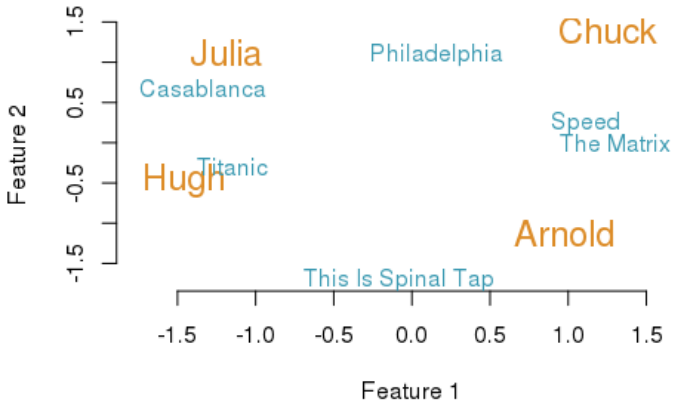


Figure 2.1: User- and item-features from truncated SVD. Note that the interpretation is not as easy as users and items that are close are a good match. Given that we only use 2 features, a user in (0, 0.01) would have a higher preference score for Philadelphia than an item in (0,0) when we model preference as the inner product of the user's and item's feature vectors.

the two leading singular values, we end up with the best rank-2 approximation, $\mathbf{US}_2\mathbf{V}^T$ of

$$\mathbf{A}_2 = \begin{pmatrix} 1.6 & 1.7 & -0.7 & -1.8 & -2.5 & 1.7 \\ -1.6 & -1.1 & 2.4 & 1.0 & -1.8 & 1.0 \\ 1.3 & 0.8 & -2.1 & -0.8 & 1.9 & -1.1 \\ -1.9 & -1.7 & 1.6 & 1.8 & 0.9 & -0.7 \end{pmatrix},$$

a reasonably good approximation to the de-meaned matrix $\mathbf{A}$. We will go into depth about the latent feature matrices in the next chapter.

In practice we often deal with matrices of substantial size, e.g. the Netflix data had 480,189 users and 17,770 movies. This gives a matrix with more than 8 billion entries, but with «only» 100 million known ratings. We will postpone until Chapter 3, the issue of how to find a SVD of a matrix with missing entries and continue with dimensionality reduction for arrays with more than 2 orders.

2.2 Tensors

2.2.1 Introduction

When working with data, the observations might have a natural representation in the form of an array, e.g. a matrix if we have users' movie ratings. Often we

have more expressive data and converting it into a matrix reduces the information. It might be of interest to know whether a user watched a movie during the day or night. In this example it could be beneficial to have a *USER*× *ITEM*× *TIME* array. A tensor is an n-dimensional array and we say that it is of order n. It functions as a multidimensional extension of matrices, where a 2-order tensor is a matrix, a 1-order array is a vector, and a 0-order array is a scalar. I will use the term tensor for arrays with $n \geq 3$. Working with tensors can be notationally cumbersome and the notation used in the literature varies.

2.2.2 Notation and Properties

I will use bold underlined capital letters to denote tensors. So the notation $\underline{\mathbf{A}} \in \mathbb{R}^{N_1 \times \ldots \times N_k}$ is used for a k-dimensional tensor, e.g. $\underline{\mathbf{A}} \in \mathbb{R}^{U \times I \times T}$ can be a tensor with U users, I items and T values of a context such as T time slices. The objective is to find a good low-dimension approximation to the true tensor $\underline{\mathbf{A}} \in \mathbb{R}^{U \times I \times T}$.

Table 2.2: Notation used in this thesis.

Symbol	How it is used
$a, \mathbf{a}, \mathbf{A}, \underline{\mathbf{A}}$	Scalar, column vector, matrix and tensor
$\|\mathbf{A}\|_F$	Frobenius norm of matrix $\mathbf{A}$
$\mathbf{a}_{\bullet i}$	A bullet indicates all indices for the subscript.
$\mathbf{x}_u \in \mathbb{R}^f$	User feature vector for $u \in \{1, 2, ..., U\}$
$\mathbf{z}_i \in \mathbb{R}^f$	Item feature vector for $i \in \{1, 2, ..., I\}$
$\mathbf{v}_o \in \mathbb{R}^f$	Context feature vector for $o \in \{1, 2, ..., T\}$
$\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n)^T \in \mathbb{R}^{U \times f}$	User feature matrix
$\mathbf{Z} = (\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_m)^T \in \mathbb{R}^{I \times f}$	Item feature matrix
$\mathbf{R} \in \mathbb{R}^{U \times I}, \underline{\mathbf{R}} \in \mathbb{R}^{U \times I \times T}$	User-Item array (duration)
$\mathbf{Y} \in \mathbb{R}^{U \times I}, \underline{\mathbf{Y}} \in \mathbb{R}^{U \times I \times T}$	Binary preference array
$\underline{\mathbf{S}} \in \mathbb{R}^{d_X \times d_Z \times d_V}$	3-order tensor in Tucker decomposition
$\mathbf{a} \circ \mathbf{b}$	Vector outer product of $\mathbf{a}, \mathbf{b} \in \mathbb{R}^a$
$\mathbf{A} \odot \mathbf{B}$	Khatri-Rao product of matrices $\mathbf{A}$ and $\mathbf{B}$
$\mathbf{A} \otimes \mathbf{B}$	Kronecker product of matrices $\mathbf{A}$ and $\mathbf{B}$
$\mathbf{A} * \mathbf{B}$	Hadamard product (elementwise matrix product)
$\underline{\mathbf{A}} \times_n \mathbf{A}$	Mode-n tensor product

Unfolding a Tensor Into a Matrix

The process of reordering the elements of a tensor into a matrix is known as *unfolding*. For the purpose of this thesis, we will only look at 3-order tensors, and only need what is known as mode-n unfolding (Kolda & Bader 2009). A mode-n unfolding of a tensor $\underline{\mathbf{A}} \in \mathbb{R}^{U \times I \times T}$ is denoted by $\mathbf{A}_{(n)}$, and rearranges the elements in the vector that results when we fix all but the n^{th} index of the tensor, and use these vectors as columns in the unfolded tensor $\mathbf{A}$. A short example

might be illustrating. Consider the tensor $\underline{\mathbf{A}} \in \mathbb{R}^{3\times1\times2}$ with

$$\mathbf{A}_{\bullet\bullet1} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \quad \mathbf{A}_{\bullet\bullet2} = \begin{pmatrix} 4 \\ 5 \\ 6 \end{pmatrix}.$$

These two slices of the third mode give the tensor $\underline{\mathbf{A}}$ in Figure 2.2.

$$\underline{\mathbf{A}} = \left[\begin{array}{cc} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{array}\right]$$

Figure 2.2: The tensor $\underline{\mathbf{A}} \in \mathbb{R}^{3\times1\times2}$ used in example of unfolding.

The three possible mode-n unfoldings of $\underline{\mathbf{A}}$ are

$$\mathbf{A}_{(1)} = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}, \quad \mathbf{A}_{(2)} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{pmatrix}, \quad \mathbf{A}_{(3)} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}.$$

For a more in-depth review of unfolding, see Kolda (2006).

Six Product Operations

The six product operations used in this thesis will be Hadamard product, inner and outer product, Kronecker product, Khatri-Rao product and n-mode tensor product. The notation is summarized in Table 2.2, but I will quickly review what the products do.

The **Hadamard product** is elementwise multiplication and will be denoted $*$. For two matrices $\mathbf{A}$ and $\mathbf{B}$, both of size $U \times I$ we have that the element

$$[\mathbf{A} * \mathbf{B}]_{ui} = a_{ui} b_{ui},$$

for $u = 1, 2, ..., U$ and $i = 1, 2, ..., I$.

The vector **inner product**, or the dot product, of $\mathbf{a} \in \mathbb{R}^f$ and $\mathbf{b} \in \mathbb{R}^f$ is sum of the elementwise multiplied vectors and is a scalar. The notation for the inner product of $\mathbf{a}$ and $\mathbf{b}$ is

$$\langle \mathbf{a}, \mathbf{b} \rangle = \mathbf{a}^T \mathbf{b}.$$

The inner product of the three vectors $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{R}^f$ is $\langle \mathbf{a}, \mathbf{b}, \mathbf{c} \rangle = \langle \mathbf{a} * \mathbf{b}, \mathbf{c} \rangle$. The vector **outer product** is denoted $\circ$. With $\mathbf{a} \in \mathbb{R}^U$ and $\mathbf{b} \in \mathbb{R}^I$ the outer product will result in a $U \times I$ matrix with elements

$$[\mathbf{a} \circ \mathbf{b}]_{ij} = [\mathbf{a}\mathbf{b}^T]_{ij} = a_i b_j.$$

We will also need the vector outer product for three vectors. Let $\mathbf{a} \in \mathbb{R}^U$, $\mathbf{b} \in \mathbb{R}^I$ and $\mathbf{c} \in \mathbb{R}^T$. The outer product of the three vectors results in a tensor $\underline{\mathbf{D}}$ of size $U \times I \times T$, with elements $d_{uit} = a_u b_i c_t$.

The **Kronecker product** $\otimes$ of two matrices $\mathbf{A} \in \mathbb{R}^{U \times I}$ and $\mathbf{B} \in \mathbb{R}^{N \times M}$ gives a $UN \times IM$ matrix.

$$\mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} a_{11}\mathbf{B} & \cdots & a_{1I}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{U1}\mathbf{B} & \cdots & a_{UI}\mathbf{B} \end{pmatrix}.$$

We can also write it expressed with the column vectors of the matrices

$$\mathbf{A} \otimes \mathbf{B} = (\mathbf{a}_1 \otimes \mathbf{b}_1 \;\; \mathbf{a}_1 \otimes \mathbf{b}_2 \;\; \ldots \;\; \mathbf{a}_I \otimes \mathbf{b}_M).$$

The fifth multiplication operation is the **Khatri-Rao product** denoted $\odot$. It is a columnwise multiplication, so given two matrices $\mathbf{A} \in \mathbb{R}^{U \times I}, \mathbf{B} \in \mathbb{R}^{N \times I}$ with equal number of columns, we have

$$\mathbf{A} \odot \mathbf{B} = (\mathbf{a}_1 \otimes \mathbf{b}_1 \;\; \mathbf{a}_2 \otimes \mathbf{b}_2 \;\; \ldots \;\; \mathbf{a}_I \otimes \mathbf{b}_I),$$

so it returns a $UN \times I$ matrix.

The last operation that will be needed is to multiply a tensor with a matrix. Taking the n**-mode product** of a tensor $\underline{\mathbf{A}} \in \mathbb{R}^{N_1 \times \ldots \times N_k}$ with a matrix $\mathbf{B} \in \mathbb{R}^{M \times N_n}$, $\underline{\mathbf{A}} \times_n \mathbf{B}$, results in a tensor of size $N_1 \times \ldots \times N_{n-1} \times M \times N_{n+1} \times \ldots \times N_k$ with elements

$$[\underline{\mathbf{A}} \times_n \mathbf{B}]_{i_1 \ldots i_{n-1} j i_{n+1} \ldots i_k} = \sum_{i_n=1}^{N_n} a_{i_1 \ldots i_k} b_{j i_n}.$$

Rank-1 Tensor

We know that a rank-1 matrix $\mathbf{A} \in \mathbb{R}^{U \times I}$ can be written as the outer product of two vectors with U and I elements. In order of emphasizing the SVD, we can also write it as

$$\mathbf{A} = \sigma(\mathbf{u} \circ \mathbf{v}), \tag{2.4}$$

where the elements are from the SVD of $\mathbf{A}$. In the same way we can write an k-way rank-1 tensor $\underline{\mathbf{A}} \in \mathbb{R}^{N_1 \times \ldots \times N_k}$ as the outer product of k vectors

$$\underline{\mathbf{A}} = \mathbf{u}_1 \circ \mathbf{u}_2 \circ \ldots \circ \mathbf{u}_k, \tag{2.5}$$

where $\mathbf{u}_i \in \mathbb{R}^{N_i}$ for $i = 1, \ldots, k$. A constant, corresponding to σ in (2.4), could be multiplied out in order to make it more similar to (2.4). A tensor's rank is

defined as the smallest number of rank-1 tensors we need to sum together to recreate the tensor. Tensor ranks are not yet well understood, and there is no straightforward algorithm that can determine the rank of a tensor (Kolda & Bader 2009). We know that the maximum rank of a $U \times I$ matrix is $\min\{U, I\}$, however the maximum of a 3-way tensor of size $U \times I \times T$ is not known, but an upper bound is given by $\min\{UI, UT, IT\}$ (Kruskal 1989). Monte Carlo simulations have shown that about 79 % of $2 \times 2 \times 2$ tensors have rank of two, while the remaining 21 % have a rank of 3 (Kruskal 1989).

2.3 Tensor Decompositions

As with matrices, we will also be interested in decomposing tensors. This can allow us to get approximations that have less noise than the original tensor, and to store good approximations with a significant reduction in memory. There are several ways to decompose a tensor, but the two most popular are the CANDECOMP/PARAFAC and the Tucker decomposition. Both have similarities to the SVD, but we will see that they both have properties that differ from SVD. For a rank F matrix $\mathbf{A}$ we can write the SVD as

$$\mathbf{A} = \mathbf{USV}^T = \sum_{r=1}^{F} \sigma_r (\mathbf{u}_r \circ \mathbf{v}_r), \tag{2.6}$$

and we know from Section 2.1.1 that the k fist terms in the sum gives the best rank-k approximation under the Frobenius norm. This result does not have an equivalent for tensors. The decompositions below are presented with equalities, but we are in most cases interested in the approximations with significantly lower upper-limits in the sums.

2.3.1 CANDECOMP/PARAFAC Decomposition

Canonical factor decomposition/Parallel factor analysis (CP) is a *linear* tensor decomposition that stems from the field of psychometrics during the 1970's. It was developed independently by Carroll & Chang (1970) and Harshman (1970). CP decomposes a tensor as a sum of rank-1 tensors. A 3-way tensor of rank-1 can be written as the outer product of three vectors as in (2.5). A given 3-way tensor $\mathbf{Y} \in \mathbb{R}^{U \times I \times T}$ of rank F can then be written as

$$\underline{\mathbf{A}} = \sum_{r=1}^{F} \mathbf{x}_r \circ \mathbf{z}_r \circ \mathbf{v}_r, \tag{2.7}$$

where $\mathbf{x}_r \in \mathbb{R}^U, \mathbf{z}_r \in \mathbb{R}^I, \mathbf{v}_r \in \mathbb{R}^T$ for $r = 1, \ldots, F$. The same size vectors are sometimes arranged together to form three matrices $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_F), \mathbf{Z} = (\mathbf{z}_1, \ldots, \mathbf{z}_F)$

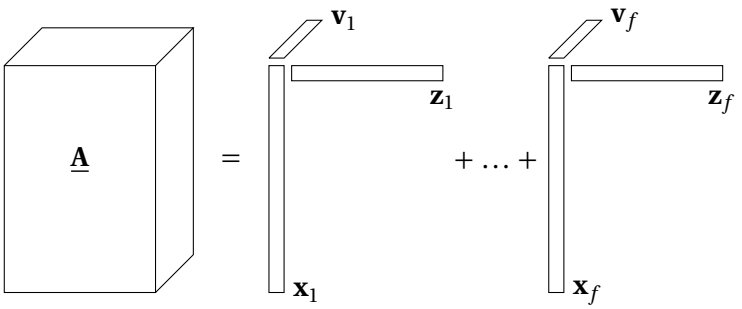


Figure 2.3: CANDECOMP/PARAFAC (CP) decomposition of a 3-way tensor with rank f. The CP decomposition factorizes the tensor into a sum of rank-1 tensors.

and $\mathbf{V} = (\mathbf{v}_1, \ldots, \mathbf{v}_F)$. This lets us write the tensor elements as

$$y_{uit} = \sum_{r=1}^{F} x_{ur} z_{ir} v_{tr}.$$

The CP decomposition is illustrated in Figure 2.3. The figure makes it easy to see that this is a 3-way tensor made up of f low rank 3-way tensors. We could easily scale out a constant σ as in (2.6), but the CP decomposition does not ensure that the vectors are orthonormal in the same way as in SVD.

One would be forgiven for believing that summing k of the low rank tensors in (2.7) would give the best k-rank approximation of the tensor $\underline{\mathbf{Y}}$. This is however generally not the case (Kolda & Bader 2009). An example of where the best CP approximation of rank-1 is not a part of the best rank-2 approximation is given in Kolda (2001). This means that we cannot solve CP approximation sequentially, but rather have to do it all at the same time. Because of the difficulties in determining the rank of a given tensor, the usual way of doing CP decomposition is to try increasing numbers of rank-1 tensors until one gets a satisfactory approximation.

We will find it useful to write $\underline{\mathbf{A}}$ using mode-n unfolding as

$$\begin{aligned}
\mathbf{A}_{(1)} &= \mathbf{X}(\mathbf{V} \odot \mathbf{Z})^T \\
\mathbf{A}_{(2)} &= \mathbf{Z}(\mathbf{V} \odot \mathbf{X})^T \\
\mathbf{A}_{(3)} &= \mathbf{V}(\mathbf{Z} \odot \mathbf{X})^T,
\end{aligned} \tag{2.8}$$

where $\mathbf{A}_{(1)}$ has size $U \times IT$, $\mathbf{A}_{(2)}$ has size $I \times UT$ and $\mathbf{A}_{(3)}$ is $T \times UI$.

2.3.2 Tucker Decomposition

Another famous tensor decomposition is the Tucker decomposition by Tucker (1966). A Tucker decomposition of a tensor $\underline{\mathbf{A}}$, decomposes the tensor into a

set of matrices and one core tensor.[1] The core matrix is multiplied with the matrices using the mode-n product. An illustration of a Tucker decomposition of a 3-way tensor $\underline{\mathbf{A}} \in \mathbb{R}^{U \times I \times T}$ can be seen in Figure 2.4. As the figure illustrates, a Tucker decomposition allows for different numbers of features for the three factors.

Before presenting the Tucker decomposition it is useful to write the $(i, j)^{\text{th}}$ element of the SVD equation (2.1) as

$$a_{ij} = \sum_a \sum_b s_{ab} u_{ia} v_{jb}. \tag{2.9}$$

This form will make it easier to see the similarities between the Tucker decomposition and SVD. In the same style we can write the elements of a 3-way tensor as

$$a_{uit} = \sum_{a=1}^{d_X} \sum_{b=1}^{d_Z} \sum_{c=1}^{d_V} s_{abc} x_{ua} z_{ib} v_{tc} \tag{2.10}$$

Note that if $d_X = d_Z = d_V$ and $\underline{\mathbf{S}}$ has ones on the diagonal entries s_{iii} and zeros elsewhere (2.10) is the same as a CP decomposition. The equality holds for the «correct» numbers d_X, d_Z, d_V. We can write the tensor as

$$\underline{\mathbf{A}} = \sum_{a=1}^{d_X} \sum_{b=1}^{d_Z} \sum_{c=1}^{d_V} s_{abc} \mathbf{x}_a \circ \mathbf{z}_b \circ \mathbf{v}_c,$$

or less cramped, with the n-mode product as

$$\underline{\mathbf{A}} = \underline{\mathbf{S}} \times_1 \mathbf{X} \times_2 \mathbf{Z} \times_3 \mathbf{V},$$

where $\underline{\mathbf{S}} \in \mathbb{R}^{d_X \times d_Z \times d_V}$, $\mathbf{X} \in \mathbb{R}^{U \times d_X}$, $\mathbf{Z} \in \mathbb{R}^{I \times d_Z}$ and $\mathbf{V} \in \mathbb{R}^{T \times d_V}$. We can see how the core tensor is multiplied with the factor matrices. The matrices $\mathbf{X}, \mathbf{Z}, \mathbf{V}$, which are often made orthonormal, will function as feature matrices. The core tensor $\underline{\mathbf{S}}$ explains the level of interaction between the different features. Since the CP decomposition is a special case of Tucker, where the core tensor is restricted, the Tucker decomposition allows for more complex interaction between features. The core describes the main relations of the full tensor (Kiers & Mechelen 2001). When the feature matrices are orthonormal the factorization is known as higher-order SVD. We can see the similarities between (2.9) and (2.10) when we consider $\mathbf{U}$, $\mathbf{Z}$ and $\mathbf{V}$ as orthonormal.

[1] In this thesis we will look at 3-way tensors and we will use Tucker decomposition for a decomposition into 3 factor matrices and a smaller core tensor. This is sometimes called a Tucker3 decomposition.

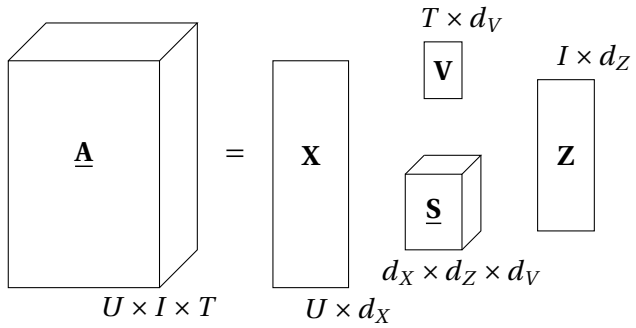


Figure 2.4: Tucker decomposition illustrated for 3-mode tensor.

We will later consider the unfolded forms of the tensor $\underline{\mathbf{A}}$. Kolda (2006) shows that the three modes we can use to unfold $\underline{\mathbf{A}}$ result in

$$\begin{aligned}
\mathbf{A}_{(1)} &= \mathbf{X}\mathbf{S}_{(1)}(\mathbf{V} \otimes \mathbf{Z})^T \\
\mathbf{A}_{(2)} &= \mathbf{Z}\mathbf{S}_{(2)}(\mathbf{V} \otimes \mathbf{X})^T \\
\mathbf{A}_{(3)} &= \mathbf{V}\mathbf{S}_{(3)}(\mathbf{Z} \otimes \mathbf{X})^T.
\end{aligned} \tag{2.11}$$

2.4 Two Optimization Algorithms

The two optimization algorithms that will be used in the next chapters will be briefly discussed here. In this section the objective will be that of minimization, but it is straightforward to switch the sign in order to maximize the function instead. The first algorithm is the very simple gradient descent algorithm of which we will use a special version called stochastic gradient descent.

2.4.1 Gradient Descent

We can find a local minimum of a function by, from an initial point, taking small steps in the negative direction of the function gradient until it converges. If the function is convex it will converge to the global minimum, otherwise it will converge to a local minimum if it exists. This first-order optimization method is called *gradient descent*. Consider an objective of the form

$$\min_{\boldsymbol{\Theta}} f(\boldsymbol{\Theta})$$

where $\boldsymbol{\Theta}$ is the set if parameters we want to optimize for. The gradient descent tells us to take a step of size α in the opposite direction of the gradient. The general outline of gradient descent is given in Algorithm 1.
A usual analogy to accompany the gradient descent is that of a blindfolded man trying to find the fastest way down a steep hill. The gradient descent way of solving this would be to stand still on one foot and use the other foot to feel

Algorithm 1 General Gradient Descent

procedure GRADIENTDESCENT(α)
  initialize:
  Assign $\boldsymbol{\Theta}$ start value
  repeat
    $\boldsymbol{\Theta}^{(k+1)} \leftarrow \boldsymbol{\Theta}^{(k)} - \alpha \nabla_{\boldsymbol{\Theta}} f(\boldsymbol{\Theta}^{(k)})$
  until convergence
  return $\boldsymbol{\Theta}$

General gradient descent algorithm for finding optimal values of $\boldsymbol{\Theta}$. The step size is given by α.

which direction the hill is steepest, then walk α steps in that direction before repeating the routine. When the blind man trying to find the bottom of the hill finds himself in a place where the hill goes up in all directions around him he will stop. Hopefully this is the global minima and not just a local minima, but with the gradient descent we are not certain to converge to the global minimum if the problem is not nice and convex.

An example of gradient descent is presented in Figure 2.5a. The function minimized is the very simple, convex function $f(x, y) = xy + x^2 + y^2$. We can see that it converges to the global minima $(0, 0)$. A special form of gradient descent is stochastic gradient descent where we only take steps for a randomly drawn parameter in each iteration.

2.4.2 Alternating Least Squares

The principles of Alternating least square (ALS) were introduced in Yates (1933). ALS is a (block) coordinate descent optimization method where one treats all but one variable as fixed and solves for the remaining. It is a well-known optimization technique and it is considered the workhorse of solving low rank CP approximations for tensors (Wright & Nocedal 1999, Wang & Navasca 2015).

ALS is very useful for instances when we can split the vector variable $\boldsymbol{\Theta}$ in the problem

$$\min_{\boldsymbol{\Theta}} f(\boldsymbol{\Theta})$$

into n sub-vectors $\boldsymbol{\Theta}_1, ..., \boldsymbol{\Theta}_n$ such that the problem can be expressed as

$$\min_{\boldsymbol{\Theta}_1, ..., \boldsymbol{\Theta}_n} f(\boldsymbol{\Theta}_1, ..., \boldsymbol{\Theta}_n),$$

where the function is convex in each $\boldsymbol{\Theta}_i$ when we treat the remaining as fixed. In coordinate descent methods such as ALS we then optimize each coordinate separately, treating the rest as fixed. The ALS algorithm is not guaranteed to reach the global minima (Kolda & Bader 2009) from a given starting point, but

it performs better or as good as six other methods in the quality of solutions to low rank decompositions in Faber et al. (2003). The starting point is often chosen randomly. We can see how ALS converges to the solution in the nice and convex problem of minimizing $f(x, y) = xy + x^2 + y^2$ in Figure 2.5b.

A highly relevant example of when we can use ALS, is with the problem of factorizing a matrix $\mathbf{A} \in \mathbb{R}^{U \times I}$ into the two lower rank matrices $\mathbf{X}$ and $\mathbf{Z}$

$$\min_{\mathbf{X} \in \mathbb{R}^{U \times f}, \mathbf{Z} \in \mathbb{R}^{I \times f}} \|\mathbf{A} - \mathbf{X}\mathbf{Z}^T\|_F^2$$

We know that the optimal solution is the truncated SVD from Section 2.1.1. Keeping $\mathbf{Z}$ fixed and starting with $\mathbf{X}$ we get the simplified problem

$$\min_{\mathbf{X} \in \mathbb{R}^{U \times f}} \|\mathbf{A} - \mathbf{X}\mathbf{Z}^T\|_F^2 = \min_{\mathbf{X} \in \mathbb{R}^{U \times f}} \operatorname{Tr}(\mathbf{A} - \mathbf{X}\mathbf{Z}^T)(\mathbf{A} - \mathbf{X}\mathbf{Z}^T)^T,$$

where $\operatorname{Tr}(\cdot)$ denotes the trace. Differentiating with respect to $\mathbf{X}$

$$\frac{\partial \operatorname{Tr}(\mathbf{A} - \mathbf{X}\mathbf{Z}^T)(\mathbf{A} - \mathbf{X}\mathbf{Z}^T)^T}{\partial \mathbf{X}} = 2\mathbf{A}\mathbf{Z} - 2\mathbf{X}\mathbf{Z}^T\mathbf{Z}$$

This gives the solution for $\mathbf{X}$ as

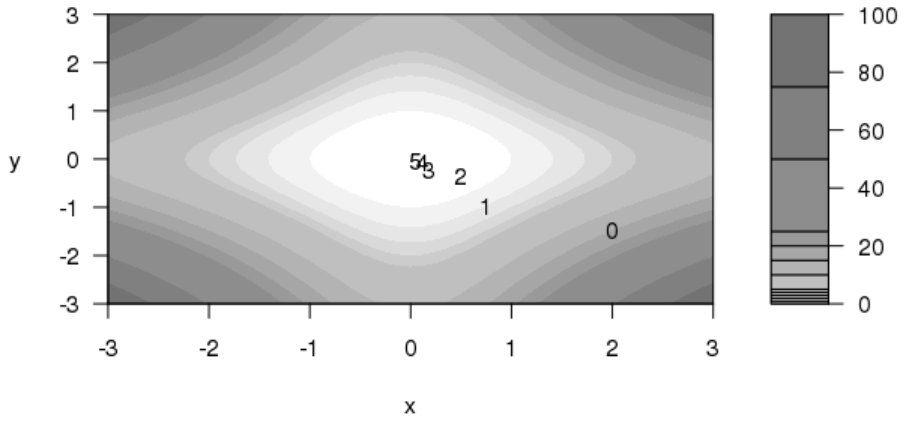
$$\mathbf{X} = \mathbf{A}\mathbf{Z}(\mathbf{Z}^T\mathbf{Z})^+ = \mathbf{A}(\mathbf{Z}^+)^T$$

Because $\mathbf{Z}^T\mathbf{Z}$ is not guaranteed to be invertible we use the pseudoinverse $(\mathbf{Z}^T\mathbf{Z})^+$. The second equality holds because $\mathbf{Z}$ can be written on reduced SVD form as $\mathbf{U}_z\mathbf{S}_z\mathbf{V}_z^T$, the pseudoinverse $\mathbf{Z}^+ = \mathbf{V}_z\mathbf{S}_z^{-1}\mathbf{U}_z^T$ and then $(\mathbf{Z}^T\mathbf{Z})^+ = \mathbf{V}_z\mathbf{S}_z^{-2}\mathbf{V}^T$. In the same way we can find that the solution for $\mathbf{Z}$, holding $\mathbf{X}$ fixed, is $\mathbf{Z} = \mathbf{A}^T(\mathbf{X}^+)^T$. Starting from a given point, ALS would thus alternate between these two solutions until they converge. The converged solution $\mathbf{X}\mathbf{Z}^T$, will be the same as the truncated SVD, since it will minimize the Frobenius norm.

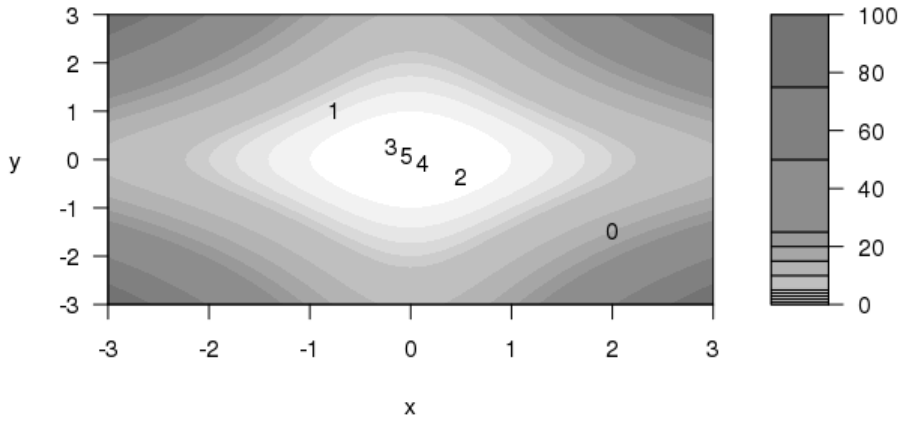
Algorithm 2 General ALS

procedure ALS
  initialize:
  Assign $\mathbf{\Theta}_1, \ldots, \mathbf{\Theta}_n$ small random numbers
  for epoch $= 1, \ldots, E$ or until convergence **do**
    for $i = 1, \ldots, n$ **do**
      Solve $f_{\mathbf{\Theta}_i}(\mathbf{\Theta}_i; \mathbf{\Theta}_1, \ldots, \mathbf{\Theta}_{i-1}, \mathbf{\Theta}_{i+1}, \ldots, \mathbf{\Theta}_n) = 0$ for $\mathbf{\Theta}_i$
  return $\mathbf{\Theta}_1, \ldots, \mathbf{\Theta}_n$

General alternating least squares algorithm for finding optimal values of $\mathbf{\Theta}_1, \ldots, \mathbf{\Theta}_n$.



(a) 5-step gradient descent with $\alpha = 0.5$



(b) 5-step alternating least squares

Figure 2.5: Visualization of two algorithms minimizing $f(x, y) = xy + x^2 + y^2$ starting in $(2, -1.5)$. (a) shows how gradient descent converges to the optimal solution (0,0), while (b) shows the same for ALS.

Chapter 3

Methods of Collaborative Filtering for Implicit Feedback

In this chapter I will introduce the collaborative filtering methods used in this thesis. All methods that will be applied to data are latent factor models, but the chapter starts with a short presentation of neighborhood models for historical reasons . The latent factor models are divided into two, context aware latent factor models in Section 3.3 and before that, context unaware models in Section 3.2. All models are based on the user - item (U-I) implicit rating matrix **R**, where each user has a row and each item has a column, the entries indicate how much the users have interacted with the items. Examples of implicit ratings are binary click data, number of interactions and the duration of interactions. In literature for explicit feedback **R** gives the known user ratings, but most of the elements are missing. The set of numbered users will be denoted $\mathscr{U} = \{1,2,...,U\}$ and the set of numbered items, mainly programs in this thesis will be denoted $\mathscr{I} = \{1,2,3,...,I\}$. The elements of $\mathscr{I}$ are on series level, i.e. each episode of a series does not get its' own entry. The preference matrix, **Y**, has elements equal to 1 if the user has interacted with an item and zero otherwise, with the assumption that interaction is a signal of preference, so

$$y_{ui} = \begin{cases} 1 & if\ r_{ui} > 0 \\ 0 & if\ r_{ui} = 0 \end{cases}$$

for all $u \in \mathscr{U}$ and $i \in \mathscr{I}$.

3.1 Neighborhood Models

Early implementations of collaborative filtering involved finding users or items with similar histories. Users who had interacted with the same items were assumed to share taste and recommendations would then be items that similar, like minded users had interacted with. One of these early implementations of CF was the netnews recommender system GroupLens (Resnick et al. 1994).

We have two alternatives for neighborhoods to consider, we can focus on finding similar users or finding similar items. This short presentation of neighborhood models will focus on the latter, item based neighborhoods, as it is considered superior on most data (Sarwar et al. 2001). Amazon is one of the companies that uses an item based neighborhood model where they recommend complementary items to the one you add to your shopping cart. About 35% of products sold on Amazon are recommended to users through recommendation algorithms.[1]

3.1.1 Item - Based Neighborhood Method

Item based neighborhood models appeared after user based, but have in many cases outperformed its older competitor. Item based neighborhood models have one very attractive quality which is that they are able to explain the recommendations in a simple way, i.e. since you have watched program i, and possibly other ones as well, we recommend movie j. Explaining why item j is recommended with something the user knows, has interacted with, instead of *similar users have enjoyed*, as would be the case with user based, can be positive.

In order to find items' neighbors we need to decide how to define how close items are to each other. One of the most famous measures of similarity is the Pearson correlation

$$corr(i,j) = \frac{\sum_{u=1}^{U}(r_{ui}-\bar{r}_i)(r_{uj}-\bar{r}_j)}{\sqrt{\sum_{u=1}^{U}(r_{ui}-\bar{r}_i)^2\sum_{u=1}^{U}(r_{uj}-\bar{r}_j)^2}},$$

between items i and j. The term $\bar{r}_i$ is the mean of the i^{th} column of $\mathbf{R}$. The other similarity measures will use values from the preference matrix $\mathbf{Y}$ instead of $\mathbf{R}$, but because Pearson correlation is often used for explicit feedback with ratings r this notation was chosen here.

For implicit feedback a common measure to use with neighborhood models is the Jaccard coefficient

$$Jacc(i,j) = \frac{\sum_{u=1}^{U} y_{ui}y_{uj}}{\sum_{u=1}^{U} y_{ui} + \sum_{u=1}^{U} y_{uj} - \sum_{u=1}^{U} y_{ui}y_{uj}}.$$

The numerator is just the total number of users who have interacted with both items. The denominator is the total number of users that have interacted with one of these items. It is simply the intersection of users who have watched program i and j divided by the union of the users who have interacted with one of the items.

[1] `http://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers`

Another popular measure of similarity is the cosine similarity

$$\cos(\theta_{ij}) = \frac{\langle \mathbf{y}_{\bullet i}, \mathbf{y}_{\bullet j} \rangle}{\|\mathbf{y}_{\bullet i}\|_F \|\mathbf{y}_{\bullet i}\|_F} = \frac{\sum_{u=1}^{U} y_{ui} y_{uj}}{\sqrt{\sum_{u=1}^{U} y_{ui}^2} \sqrt{\sum_{u=1}^{U} y_{uj}^2}} \tag{3.1}$$

where θ_{ij} is the angle between columns i and j in the preference matrix $\mathbf{Y}$. The cosine similarity takes values from -1, indicating exactly opposite, to 1, indicating exactly equal.

Now that we can find out how similar items are, we can either recommend the items that are closest to the item the user last interacted with, or we can do a weighted sum of the k last items the user interacted with and aggregate the results. While item based neighborhood models seem to perform better than user based ones, they are both usually outperformed by matrix factorization models(Rendle & Schmidt-Thieme 2010). We will leave neighborhood models and just briefly revisit the cosine similarity in Chapter 5.

3.2 Latent Factor Models

While neighborhood models might have been popular, the Netflix prize showed that it was usually outperformed by simple latent factor models (Funk 2006). With latent factor models users and items are mapped to the same latent factor space. The idea behind is that users' taste can be modeled by just a few latent factors and thus, that we do not need to consider the user's complete history when we predict if he will enjoy an item he has not interacted with. These features could be anything from degree of action to the presence of religious figures, or a more blurry mix. A user's taste is then a vector where each element is the degree of how much he likes that feature in the item. And correspondingly an item will have a vector where the elements show how much of each latent feature it has. The preference score is then the inner product of the latent feature vectors

$$\hat{y}_{ui} = \langle \mathbf{x}_u, \mathbf{z}_i \rangle,$$

where $\mathbf{x}_u \in \mathbb{R}^f$ and $\mathbf{z}_i \in \mathbb{R}^f$ are the latent feature vector for user u and item i. We want the inner product of the feature vectors to represent how well the user and item *agrees* with each other, or really how much the user would enjoy the item. This is done for all users and items and so, we can represent the predicted enjoyment of all users for all items as $\mathbf{X}\mathbf{Z}^T \in \mathbb{R}^{U \times I}$, where we use the feature matrices $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_U)^T$ and $\mathbf{Z} = (\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_I)^T$. So we take the preference matrix $\mathbf{Y}$ and factorize it into a user feature matrix $\mathbf{X}$ and an item feature matrix $\mathbf{Z}$. The goal is to find an approximation $\mathbf{X}\mathbf{Z}^T$ that minimizes a loss function $L(\mathbf{X}, \mathbf{Z}|\mathbf{Y})$. Latent factor models where we factorize the preference matrix, or

rating matrix, into one latent feature matrix for users and one for items, go under the name matrix factorization models. Matrix factorization (MF) is the task of factorizing a matrix into the product of matrices. This thesis will focus on such latent feature models, and different matrix factorization models will be presented below.

The training time with latent factor models are often significant, but when the model is learned making all predictions is as simple as one matrix product. This thesis does not go into details about the training times of the different models. The recommendations for each user are the N items the user has highest preference score with, that he has not already interacted with. The first algorithm to be presented is the simplest one, namely Binary regularized matrix factorization. It is very closely related to the method with truncated SVD showed in Section 2.1.

3.2.1 Binary Regularized Matrix Factorization

As the name of this section suggests, we will work with the binary preference matrix $\mathbf{Y}$, where an matrix entry equal to 1 signals appreciation and a 0 signals dislike. We want to model the preference with a preference score that is the inner product of the user's and item's feature vectors $\mathbf{x}_u, \mathbf{z}_i \in \mathbb{R}^f$. So the preference score is

$$y_{ui} \approx \langle \mathbf{x}_u, \mathbf{z}_i \rangle.$$

We want it to be high if user u enjoys i and low otherwise. So one part of the loss function is to approximate $\mathbf{X}\mathbf{Z}^T$ to $\mathbf{Y}$ by minimizing

$$\|\mathbf{Y} - \mathbf{X}\mathbf{Z}^T\|_F^2, \tag{3.2}$$

the sum of squared errors. We want to minimize (3.2) with respect to $\mathbf{X} \in \mathbb{R}^{U \times f}$ and $\mathbf{Z} \in \mathbb{R}^{I \times f}$. We note that this is the same as finding the best truncated SVD as shown in Section 2.1.1. Although this is exactly what will be done in Section 5.3.4, this approach has several weaknesses. The main one being that it is prone to overfitting, i.e. having small training error but large error on new data. To prevent overfitting it is custom to add a regularization term, that for mathematical reasons, will be the l_2-penalty. We will dwell deeper into regularization and overfitting in the next section. The loss function we minimize in the Binary regularized matrix factorization model is

$$L(\mathbf{X}, \mathbf{Z}) = \sum_{u,i} (y_{ui} - \mathbf{x}_u^T \mathbf{z}_i)^2 + \lambda \left(\sum_u \|\mathbf{x}_u\|_F^2 + \sum_i \|\mathbf{z}_i\|_F^2 \right)$$
$$s.t. \quad \mathbf{X} \in \mathbb{R}^{U \times f}, \mathbf{Z} \in \mathbb{R}^{I \times f}. \tag{3.3}$$

Here λ is the weight of the l_2-penalty and a tuning parameter. With explicit feedback, the traditional way of solving the problem of missing entries in $\mathbf{R}$, is

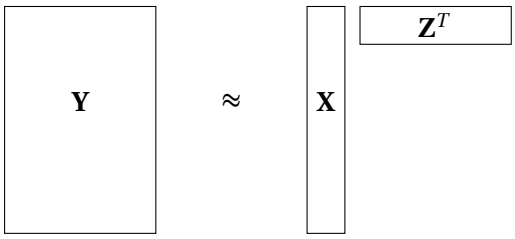


Figure 3.1: Factorizing $\mathbf{Y} \in \mathbb{R}^{U \times I}$ into a user feature matrix $\mathbf{X} \in \mathbb{R}^{U \times f}$ and an item feature matrix $\mathbf{Z} \in \mathbb{R}^{I \times f}$ is the objective of matrix factorization.

by only minimizing over the user-item pairs where we know the ratings. Solving (3.3) for $(\mathbf{X}, \mathbf{Z})$ is very similar to the matrix factorization in Section 2.4. It will also be done for the more general problem in Section 3.2.3, where we solve for each row of the feature matrices at a time. Since the number of latent features is f we have $f(U+I)$ features to compute. The number of latent features should be chosen as a compromise between training time and prediction accuracy. The regularization parameter λ is usually chosen from cross validation or another validation scheme. We can rewrite the loss function in (3.3) on matrix form

$$\begin{aligned} L(\mathbf{X}, \mathbf{Z}) = \|\mathbf{Y} - \mathbf{X}\mathbf{Z}^T\|_F^2 + \lambda\left(\|\mathbf{X}\|_F^2 + \|\mathbf{Z}\|_F^2\right) \\ s.t. \quad \mathbf{X} \in \mathbb{R}^{U \times f}, \, \mathbf{Z} \in \mathbb{R}^{I \times f}. \end{aligned}$$

The objective of minimizing the sum of squared errors like we do with Binary regularized matrix factorization also has a probabilistic interpretation. If we assume that y_{ui} is normally distributed with expectation given by the inner product of the latent feature vectors we will see that we can get to the same optimization problem. The normality assumption may not seem very reasonable with observations with values of only zero and one, but how much the interaction between user and item agrees could be. If we let y_{ui} be $\mathcal{N}(\mathbf{x}_u^T \mathbf{z}_i, \sigma^2)$ with variance equal for all y, we have the likelihood or conditional distribution

$$\begin{aligned} Lik(\mathbf{Y}|\mathbf{X}, \mathbf{Z}, \alpha) &= \prod_{u,i} \left[\mathcal{N}(y_{ij}|\mu = \mathbf{x}_u^T \mathbf{z}_i, \sigma^2)\right] \\ &= \prod_{u,i} \left[\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(y_{ui} - \mathbf{x}_u^T \mathbf{z}_i)^2\right)\right]. \end{aligned}$$

With the traditional monotone transformation using the natural logarithm, we get the log-likelihood proportional to

$$l(\mathbf{X}, \mathbf{Z}) \propto -\sum_{u,i} \left(y_{ui} - \mathbf{x}_u^T \mathbf{z}_i\right)^2.$$

Maximizing this is equivalent to minimizing the Frobenius norm, $\|\mathbf{Y} - \mathbf{X}\mathbf{Z}^T\|_F$, as shown in Section 2.1.1. Moreover, if we introduce priors on the feature matrices,

such that each row of $\mathbf{X}$ is Gaussian $\mathbf{x}_u \sim \mathcal{N}_f(\mathbf{0}, \frac{\sigma^2}{\lambda}\mathbf{I})$ for $u \in \mathcal{U}$. And likewise for the item feature vectors $\mathbf{z}_i \sim \mathcal{N}_f(\mathbf{0}, \frac{\sigma^2}{\lambda}\mathbf{I})$ and all elements are independent, we get

$$l(\mathbf{X}, \mathbf{Z}) \propto -\sum_{u,i} \left(y_{ui} - \mathbf{x}_u^T \mathbf{z}_i\right)^2 - \lambda \left(\sum_u \|\mathbf{x}_u\|_F^2 + \sum_i \|\mathbf{z}_i\|_F^2 \right).$$

This is exactly the same task as we had with the Binary regularized matrix factorization. This is a simple example of what is known as probabilistic matrix factorization (Salakhutdinov & Mnih 2007).

The BRMF model treats two items that the user has interacted with the same even if he just started one and has watched the other several times. Watching several episodes of a series or a movie more than once is something you would do if you found it interesting. This is a much stronger signal that the user likes the item than that the user only started watching the program. So these entries should be more important to get correctly in the predicted matrix. Weighted regularized matrix factorization, to be presented in Section 3.2.3, has a clever way of ensuring this.

3.2.2 Regularization and Overfitting

When doing matrix factorization it is important to be aware of the non-uniqueness of the factorization

$$\mathbf{Y} = \mathbf{X}\mathbf{Z}^T. \tag{3.4}$$

For any invertible $f \times f$ matrix $\mathbf{G}$ we have

$$\mathbf{Y} = \mathbf{X}\mathbf{Z}^T = \mathbf{X}\mathbf{G}\mathbf{G}^{-1}\mathbf{Z}^T = \tilde{\mathbf{X}}\tilde{\mathbf{Z}}^T,$$

where $\tilde{\mathbf{X}} = \mathbf{X}\mathbf{G}$ and $\tilde{\mathbf{Z}}^T = \mathbf{G}^{-1}\mathbf{Z}$. Equation (3.4) shows that there are infinitely many solutions to the decomposition $\mathbf{Y} = \mathbf{X}\mathbf{Z}^T$. Multiplying $\mathbf{X}$ with a matrix $\mathbf{G}$ can cause matrix $\tilde{\mathbf{X}}$ to be differently scaled and/or rotated compared to $\mathbf{X}$. We can control the issue of scaling with regularization, as recommended in Buchanan & Fitzgibbon (2005). With MF there are a lot of matrix entries to estimate, often with few events, and the regularization parameter forces these elements towards zero. This helps prevent overfitting as users and items with few events will too easy to fit to their positive entries and can end up with some extreme values in their feature vectors. This becomes a big problem when we later do predictions. Another cause for overfitting is to choose a too high number of features f. This can give a saturated model that would be unable to give recommendations, since it would be able to predict 0s for all the items the user has not already interacted with. However, Salakhutdinov & Mnih (2007) demonstrated that overfitting is a problem for MF even for low numbers of latent features. Adding

regularization to the loss function we minimize, thus reduces the problem of scaling and overfitting. The issue of rotation is not problematic as we have no interest in interpreting the features and the rotated matrices give the same explanatory power.

The regularization term, λ, will consistently be equal for all latent feature vectors. This means that the same λ will be used to penalize entries in $\mathbf{X}$ and $\mathbf{Z}$. This is done because of (i) readability, (ii) there is no principle difference for users' and items' latent features that necessitates it and (iii) it will be less tuning parameters to choose when implementing the models.

3.2.3 Weighted Regularized Matrix Factorization

The Weighted regularized matrix factorization (WRMF) model in Hu et al. (2008) is quite similar to the Binary regularized matrix factorization. The score is still the inner product of the user and item feature vectors. The difference is that we let the user - item interactions with the highest interaction values r, be more important to fit in the loss function. The basic idea is that how much you like the item is positively correlated with how much you interact with it. We are more sure that a user enjoys an item he has interacted with several times, than that he does not like an item he has not interacted with. So if you watch a movie to the end three times we would be more confident that you like it, compared to a movie you only saw 15 minutes of once. On the other hand the model treats unobserved as negative with low confidence. Events with high confidence will be more valuable when we estimate the users' and items' latent features. We care more that the preference scores for these events are close to the observed value compared to events with low confidence. This seems reasonable and the method has enjoyed great success. It has become more or less the go-to method for implicit feedback data, and it is currently[2] the only CF model NRK bases its recommendations to users on.

While we still find user u's score for item i by $\hat{y}_{ui} = \langle \mathbf{x}_u, \mathbf{z}_i \rangle$, the WRMF model introduces a confidence function that increases with duration. In this thesis I will use the simple function

$$c = c(r) = 1 + \alpha r, \tag{3.5}$$

where α is a positive parameter. The constant 1 is the confidence we give the negative observations, i.e. where we do not have interactions. So the observed negatives, the entries for combination of users and items with no events, have low confidence which is reasonable as we do not know which are from dislike and which are from unawareness. The tuning parameter α is chosen from a

[2] As of 14, May 2017.

validation method such as cross validation. The loss function we minimize for WRMF is

$$L(\mathbf{X},\mathbf{Z}) = \sum_{u,i} c_{ui}(y_{ui} - \mathbf{x}_u^T \mathbf{z}_i)^2 + \lambda \left(\sum_u \|\mathbf{x}_u\|_F^2 + \sum_i \|\mathbf{z}_i\|_F^2 \right)$$
$$s.t. \quad \mathbf{X} \in \mathbb{R}^{U \times f}, \mathbf{Z} \in \mathbb{R}^{I \times f}. \tag{3.6}$$

Note that with $c_{ui} = 1$ for every combination of user and item, corresponding to $\alpha = 0$ in (3.5), gives the BRMF. Pan et al. (2008) recommends using $c = 1$ for combinations of u and i where $y_{ui} = 1$, and a lower value for the negatives where $y_{ui} = 0$. We will solve (3.6) numerically by alternating least squares. The computations will be presented in the following section.

Alternating Least Squares for Weighted Regularized Matrix Factorization

Solving (3.6) for $\mathbf{x}$ and $\mathbf{z}$ simultaneously is challenging as the problem is non-convex. However, if we choose to solve for one of them, while holding the other fixed, the problem becomes nice and convex. ALS was presented in Section 2.4.2 and we can use it for $\mathbf{x}\,\mathbf{z}$, one at the time. We then optimize

$$f(\mathbf{x}_u) = (\mathbf{y}_{u\bullet} - \mathbf{Z}\mathbf{x}_u)^T \mathbf{C}_u (\mathbf{y}_{u\bullet} - \mathbf{Z}\mathbf{x}_u) + \lambda \mathbf{x}_u^T \mathbf{x}_u \tag{3.7}$$
$$f(\mathbf{z}_i) = (\mathbf{y}_{\bullet i} - \mathbf{X}\mathbf{z}_i)^T \mathbf{C}_i (\mathbf{y}_{\bullet i} - \mathbf{X}\mathbf{z}_i) + \lambda \mathbf{z}_i^T \mathbf{z}_i. \tag{3.8}$$

The matrix $\mathbf{C}_u$ in (3.7) is a matrix with the elements $\mathbf{c}_{u\bullet}$ as the diagonal. The similarly named matrix $\mathbf{C}_i$ in (3.8) is diagonal with the elements $\mathbf{c}_{\bullet i}$. Note that both $\mathbf{y}_{u\bullet} \in \mathbb{R}^U$ and $\mathbf{y}_{\bullet i} \in \mathbb{R}^U$ are column vectors. We will see that (3.7) and (3.8) have nice analytic solutions. The first order conditions for the ALS are

$$\frac{\partial f(\mathbf{x}_u)}{\partial \mathbf{x}_u} = -2\mathbf{Z}^T \mathbf{C}_u \mathbf{y}_{u\bullet} + 2\mathbf{Z}^T \mathbf{C}_u \mathbf{Z}\mathbf{x}_u + 2\lambda \mathbf{x}_u = 0$$
$$\frac{\partial f(\mathbf{z}_i)}{\partial \mathbf{z}_i} = -2\mathbf{X}^T \mathbf{C}_i \mathbf{y}_{\bullet i} + 2\mathbf{X}^T \mathbf{C}_i \mathbf{X}\mathbf{z}_i + 2\lambda \mathbf{z}_i = 0.$$

Solving these first order conditions for the unfixed variables gives the closed form solutions

$$\mathbf{x}_u = (\mathbf{Z}^T \mathbf{C}_u \mathbf{Z} + \lambda \mathbf{I})^{-1} \mathbf{Z}^T \mathbf{C}_u \mathbf{y}_{u\bullet}$$
$$\mathbf{z}_i = (\mathbf{X}^T \mathbf{C}_i \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{C}_i \mathbf{y}_{\bullet i}$$

Often it is computationally faster not to find the inverted matrix, but rather use a linear equation solver on the equations

$$(\mathbf{Z}^T \mathbf{C}_u \mathbf{Z} + \lambda \mathbf{I})\mathbf{x}_u = \mathbf{Z}^T \mathbf{C}_u \mathbf{y}_{u\bullet}$$
$$(\mathbf{X}^T \mathbf{C}_i \mathbf{X} + \lambda \mathbf{I})\mathbf{z}_i^T = \mathbf{X}^T \mathbf{C}_i \mathbf{y}_{\bullet i}.$$

Pseudocode for the ALS algorithm for WRMF is shown in Algorithm 3.

Algorithm 3 Alternating least squares for WRMF

procedure ALSWRMF($\mathbf{C}, \mathbf{Y}, E, \lambda$)
  initialize:
  $\mathbf{X} \in \mathbb{R}^{U \times f}, \mathbf{Z} \in \mathbb{R}^{I \times f}$ with small random numbers
  for epoch $= 1, \ldots, E$ **do**
    for $u = 1, \ldots, U$ **do**
      $\mathbf{C}_u = \text{diag}(\mathbf{c}_{u\bullet})$
      $\mathbf{x}_u = (\mathbf{Z}^T \mathbf{C}_u \mathbf{Z} + \lambda \mathbf{I})^{-1} \mathbf{Z}^T \mathbf{C}_u \mathbf{y}_{u\bullet}$
      for $i = 1, \ldots, I$ **do**
        $\mathbf{C}_i = \text{diag}(\mathbf{c}_{\bullet i})$
        $\mathbf{z}_i = (\mathbf{X}^T \mathbf{C}_i \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{C}_i \mathbf{y}_{\bullet i}$
  return $\mathbf{X}, \mathbf{Z}$

Algorithm for finding optimal values of $\mathbf{X}$ and $\mathbf{Z}$ in the Weighted regularized matrix factorization model.

3.2.4 Bayesian Personalized Ranking

Even though WRMF might be the most used method of doing collaborative filtering for implicit feedback, it is not really optimized for personalized ranking. With WRMF the model is trained treating the unknowns we want to rank as negatives. Rendle et al. (2009) argue that for ranking it is more appropriate to use pairwise comparison than the pointwise fitting of WRMF. The unknown functions as a negative only directly compared to a known positive. They introduce the Bayesian personalized ranking optimality criterion: BPR-Opt. Although BPR-Opt is a general criterion I will only present it in the special case where we use matrix factorization in the score function.

The idea is that we instead of fitting one entry at the time, pointwise, we do a pairwise comparison with one positive and one negative entry. Because we always compare two items, one that the user has interacted with and one that he has not, we can create the set

$$\mathscr{D}_u = \left\{(i, j) : i \in \mathscr{I}_u^+,\ j \in \mathscr{I} \setminus \mathscr{I}_u^+\right\}$$

for all $u \in \mathscr{U}$. $\mathscr{I}_u^+$ is the set of items that user u has interacted with and thus item j is an item the user has not interacted with. The set gives, for every user, all combinations of item pairs where the user has interacted with the first, but not the last. The assumption is that the user prefers the items he has interacted with to the ones he has not interacted with. We still model the score as the inner product of the user's and item's latent feature vectors

$$\hat{y}_{ui} = \langle \mathbf{x}_u, \mathbf{z}_i \rangle.$$

For simplicity, let us introduce the notation

$$\hat{y}_{uij} = \hat{y}_{ui} - \hat{y}_{uj},$$

the difference in the preference score. We model the probability that user u prefers item i to item j, i.e. $i \succ_u j$, as

$$p(i \succ_u j | \mathbf{x}_u, \mathbf{z}_i, \mathbf{z}_j) = \frac{1}{1+e^{-\hat{y}_{uij}}}.$$

This is the logistic function and it is the one used in Rendle et al. (2009), but we could choose another probability function that is increasing in $\hat{y}_{uij}$. If we assume that all pairwise comparisons are independent given the feature vectors, we get the user's likelihood

$$p(\succ_u | \mathbf{x}_u, \mathbf{Z}) = \prod_{(i,j) \in \mathscr{D}_u} \frac{1}{1+e^{-\hat{y}_{uij}}}.$$

This is the probability that the user really prefers all items he has watched to all else, given the features. With independence between users we get the total conditional likelihood for all users becomes

$$\begin{aligned} p(\succ_{\mathscr{U}} | \mathbf{X}, \mathbf{Z}) &= \prod_{u \in \mathscr{U}} p(\succ_u | \mathbf{x}_u, \mathbf{Z}) \\ &= \prod_{u \in \mathscr{U}} \prod_{(i,j) \in \mathscr{D}_u} \frac{1}{1+e^{-\hat{y}_{uij}}} \\ &= \prod_{(u,i,j) \in \mathscr{D}} \frac{1}{1+e^{-\hat{y}_{uij}}}, \end{aligned}$$

where now $\mathscr{D}$ is the set with triplets (u,i,j) such that user u is assumed to prefer i to j. What we are interested in is the posterior of the variables $\mathbf{X}$ and $\mathbf{Z}$. If we assume that the rows of $\mathbf{X}$ and $\mathbf{Z}$ are independent with centered Gaussian priors with the same diagonal covariance matrix of the form $\frac{2}{\lambda}\mathbf{I}$, we know from Bayesian inference that the posterior has the relation

$$p(\mathbf{X}, \mathbf{Z} | \succ_{\mathscr{U}}) \propto \left(\prod_{(u,i,j) \in \mathscr{D}} \frac{1}{1+e^{-\hat{y}_{uij}}} \right) \left(\prod_{u \in \mathscr{U}} e^{-\frac{\lambda}{2}\mathbf{x}_u^T \mathbf{x}_u} \right) \left(\prod_{i \in \mathscr{I}} e^{-\frac{\lambda}{2}\mathbf{z}_i^T \mathbf{z}_i} \right). \tag{3.9}$$

The problem of maximizing (3.9) is equivalent to maximizing the simpler

$$\ln p(\mathbf{X}, \mathbf{Z} | \succ_{\mathscr{U}}) \propto \sum_{(u,i,j) \in \mathscr{D}} \ln \frac{1}{1+e^{-\hat{y}_{uij}}} - \frac{\lambda}{2} \left(\|\mathbf{X}\|_F^2 + \|\mathbf{Z}\|_F^2 \right), \tag{3.10}$$

which is the BPR-Opt criterion. We will use the name Bayesian personalized ranking with matrix factorization, BPR with MF for short, for the model that maximizes (3.10). To maximize (3.10) we will use stochastic gradient descent, where we draw one triplet $(u,i,v) \in \mathscr{D}$ and do gradient descent. We need to find the derivatives of the variables involved for one triplet $(u,i,j) \in \mathscr{D}$

$$\sum_{(u,i,j) \in \mathscr{D}} f_{uij}(\mathbf{x}_u, \mathbf{z}_i, \mathbf{z}_j) = \sum_{(u,i,j) \in \mathscr{D}} \left\{ \ln \frac{1}{1+e^{-\hat{y}_{uij}}} - \frac{\lambda}{2} \left(\mathbf{x}_u^T \mathbf{x}_u + \mathbf{z}_i^T \mathbf{z}_i + \mathbf{z}_j^T \mathbf{z}_j \right) \right\}.$$

Let (u, i, j) be the triplet drawn in one of the draws. Then the relevant derivatives are

$$\frac{\partial f_{uij}(\mathbf{x}_u, \mathbf{z}_i, \mathbf{z}_j)}{\partial \mathbf{x}_u} = \frac{1}{1+e^{\hat{y}_{uij}}}(\mathbf{z}_i - \mathbf{z}_j) - \lambda \mathbf{x}_u$$
$$\frac{\partial f_{uij}(\mathbf{x}_u, \mathbf{z}_i, \mathbf{z}_j)}{\partial \mathbf{z}_i} = \frac{1}{1+e^{\hat{y}_{uij}}}\mathbf{x}_u - \lambda \mathbf{z}_i$$
$$\frac{\partial f_{uij}(\mathbf{x}_u, \mathbf{z}_i, \mathbf{z}_j)}{\partial \mathbf{z}_j} = -\frac{1}{1+e^{\hat{y}_{uij}}}\mathbf{x}_u - \lambda \mathbf{z}_j.$$

The derivatives above are all we need to know to do stochastic gradient descent and the pseudocode is given in Algorithm 4. An obvious drawback of BPR with MF is that it does not include the implicit rating for the pairwise ranking, it treats them as binary in the same way as $\mathbf{Y}$. Instances where we have two items that a user has interacted with are not ranked against each other directly even if one would have r-value of magnitudes larger than the other. The biggest advantage is that it uses a logistic function that is more appropriate for the classification task that recommendations of top-N lists really are.

Algorithm 4 Stochastic Gradient Descent for Bayesian personalized ranking

procedure LEARNBPRMF($\mathscr{D}$)
  initialize:
  $\mathbf{X} \in \mathbb{R}^{U \times f}, \mathbf{Z} \in \mathbb{R}^{I \times f}$ with small random numbers
  repeat
    draw $(u, i, j) \in \mathscr{D}$
    $\mathbf{x}_u = \mathbf{x}_u + \alpha\left(\frac{1}{1+e^{\hat{y}_{uij}}}(\mathbf{z}_i - \mathbf{z}_j) - \lambda \mathbf{x}_u\right)$
    $\mathbf{z}_i = \mathbf{z}_i + \alpha\left(\frac{1}{1+e^{\hat{y}_{uij}}}\mathbf{x}_u - \lambda \mathbf{z}_i\right)$
    $\mathbf{z}_j = \mathbf{z}_j + \alpha\left(-\frac{1}{1+e^{\hat{y}_{uij}}}\mathbf{x}_u - \lambda \mathbf{z}_j\right)$
  until convergence
  return $\mathbf{X}, \mathbf{Z}$

Algorithm for finding optimal values of $\mathbf{X}, \mathbf{Z}$ for matrix factorization with Bayesian personalized ranking criterion. The scalar α is the learning rate.

3.2.5 Logistic Matrix Factorization

Another model for implicit feedback we solve by gradient descent is Spotify's Logistic matrix factorization (Johnson 2014). This model will not be applied to data, but it is another take on MF with implicit feedback that is easy to grasp. Johnson (2014) model the probability that a user will interact with an item as a logistic function

$$p(y_{ui} = 1 | \mathbf{x}_u, \mathbf{z}_i, \beta_u, \beta_i) = \frac{e^{\mathbf{x}_u^T \mathbf{z}_i + \beta_u + \beta_i}}{1 + e^{\mathbf{x}_u^T \mathbf{z}_i + \beta_u + \beta_i}},$$

where $y_{ui} = 1$ if the user u interacts with item i and 0 otherwise, $\mathbf{x}_u \in \mathbb{R}^f$, $\mathbf{z}_i \in \mathbb{R}^f$ are latent feature vectors and β_u, β_i are user and item biases. Biases allows for some users and items to have a higher probability of having events. Assuming all observations, given feature vectors and biases, are independent, we get the likelihood or conditional distribution

$$L(\mathbf{Y}|\mathbf{X},\mathbf{Z},\beta) = \prod_{u,i} p(y_{ui}=1|\mathbf{x}_u,\mathbf{z}_i,\beta_u,\beta_i)^{\alpha r_{ui}}(1-p(y_{ui}=1|\mathbf{x}_u,\mathbf{z}_i,\beta_u,\beta_i)),$$

where r_{ui}, for Spotify, is how many times user u has listened to song i. We see that the tuning parameter α acts as a weight, much in the same way as with the confidence in WRMF of Section 3.2.3. If we apply centered Gaussian priors for $\mathbf{X}$ and $\mathbf{Z}$, as was done in Section 3.2.1, we get the posterior distribution we want to maximize

$$\begin{aligned} p(\mathbf{X},\mathbf{Z},\beta|\mathbf{Y},\sigma_X\mathbf{I},\sigma_Z\mathbf{I}) &\propto p(\mathbf{X}|\sigma_X^2\mathbf{I})p(\mathbf{Z}|\sigma_Z^2\mathbf{I})L(\mathbf{Y}|\mathbf{X},\mathbf{Z},\beta) \\ &\propto e^{-\frac{\lambda}{2}\mathbf{X}^T\mathbf{X}}e^{-\frac{\lambda}{2}\mathbf{Z}^T\mathbf{Z}}\prod_{u,i}\left(\frac{e^{\mathbf{x}_u^T\mathbf{z}_i+\beta_u+\beta_i}}{1+e^{\mathbf{x}_u^T\mathbf{z}_i+\beta_u+\beta_i}}\right)^{\alpha r_{ui}}\left(\frac{1}{1+e^{\mathbf{x}_u^T\mathbf{z}_i+\beta_u+\beta_i}}\right). \end{aligned} \tag{3.11}$$

Here the notation is abused. $\mathbf{X}$ and $\mathbf{Z}$ are vectorized versions of the matrices. The problem of maximizing (3.11) is equivalent to

$$\begin{aligned} \max_{\mathbf{X},\mathbf{Z},\beta}\Bigg\{\sum_{u,i}\Big(\alpha r_{ui}(\mathbf{x}_u^T\mathbf{z}_i+\beta_u+\beta_i)-(1+\alpha r_{ui})\ln(1+e^{\mathbf{x}_u^T\mathbf{z}_i+\beta_u+\beta_i})\Big) \\ -\frac{\lambda}{2}\sum_u\|\mathbf{x}_u\|_F^2-\frac{\lambda}{2}\sum_i\|\mathbf{z}_i\|_F^2\Bigg\}. \end{aligned} \tag{3.12}$$

This is not difficult to solve this with gradient descent, but as this model will not be applied to data, it will not be pursued here.

3.2.6 Context Aware Matrix Factorization

A way of adding context to RS is to train different models for each level of context. By training separate models for different contexts the different models will not be able to borrow strength from each other. Thus if users do not change their preferences randomly given a new context, we would assume that a model that can borrow strength over contexts will perform better/learn the latent factors more precisely.

3.3 Tensor Factorization Models

In order to incorporate context variables in the model, and inspired by the success of MF, factorization of higher order arrays was applied in Karatzoglou

et al. (2010). Tensor-based models are today considered to be state of the art of context-aware recommender systems. Using an array with 3 dimensions we can model the interaction between the 3 entities user, item and context category. The tensors $\underline{\mathbf{R}}$ and $\underline{\mathbf{Y}}$ are created in the same fashion as the corresponding matrices. The entry r_{uit} is user us implicit rating of item i under context t, and so on. Adding a context dimension allows a user's program preferences to change depending on the context category. Tensor decomposition models are very similar to MF models as will be seen in this part.

3.3.1 Weighted Regularized CP Factorization

Hidasi & Tikk (2012) presented a severely modified version of the WRMF model that is able to take context into account through a CP tensor decomposition. The model reviewed here is less optimized than Hidasi & Tikk (2012). The notation also differs substantially from what is used here. We will model the preference of a user u for item i under context t, y_{uit}, as a 3-way CP factorization where each user, item and context category has a latent feature vector. Remember that CP decomposition is a decomposition into a sum of rank-1 tensors where element y_{uit} can be written as

$$\hat{y}_{uit} = \sum_{r=1}^{f} x_{ur} z_{ir} v_{tr} = \langle \mathbf{x}_u, \mathbf{z}_i, \mathbf{v}_t \rangle = (\mathbf{x}_u * \mathbf{z}_i * \mathbf{v}_t)^T \mathbf{1}.$$

The feature vectors $\mathbf{x}_u, \mathbf{z}_i, \mathbf{v}_t$ each have f elements and $\mathbf{1}$ is column vector with f elements, all equal to 1. We can stack the vectors in the same way as under MF, such that $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_U)^T$, $\mathbf{Z} = (\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_I)^T$, but now we also have the context feature matrix $\mathbf{V} = (\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_T)^T$. The tensor model will allow for users item preferences to differ under different context levels.

In the same spirit as WRMF we have the loss function

$$L(\mathbf{X}, \mathbf{Z}, \mathbf{V}) = \sum_{u,i,t} c_{uit} \left(y_{uit} - (\mathbf{x}_u * \mathbf{z}_i * \mathbf{v}_t)^T \mathbf{1} \right)^2 + \lambda \left(\sum_u \|\mathbf{x}_u\|_F^2 + \sum_i \|\mathbf{z}_i\|_F^2 + \sum_t \|\mathbf{v}_t\|_F^2 \right). \quad (3.13)$$

Fixing (3.13) for two of $\mathbf{x}_u, \mathbf{z}_i$ and $\mathbf{v}_t$, we have a problem that is convex in the unfixed feature vector and we can solve it with ALS. We get the first order deriva-

tives

$$\frac{\partial L(\mathbf{X},\mathbf{Z},\mathbf{V})}{\partial \mathbf{x}_v} = -2\sum_{i,t} c_{vit}\, y_{vit}(\mathbf{z}_i * \mathbf{v}_t)^T + 2\sum_{i,t} c_{vit}(\mathbf{z}_i^2 * \mathbf{v}_t^2)^T * \mathbf{x}_v^T + 2\lambda \mathbf{x}_v^T \tag{3.14}$$

$$\frac{\partial L(\mathbf{Z},\mathbf{X},\mathbf{V})}{\partial \mathbf{z}_j} = -2\sum_{u,t} c_{ujt}\, y_{ujt}(\mathbf{x}_u * \mathbf{v}_t)^T + 2\sum_{u,t} c_{ujt}(\mathbf{x}_u^2 * \mathbf{v}_t^2)^T * \mathbf{z}_j^T + 2\lambda \mathbf{z}_j^T \tag{3.15}$$

$$\frac{\partial L(\mathbf{V},\mathbf{X},\mathbf{Z})}{\partial \mathbf{v}_s} = -2\sum_{u,i} c_{uis}\, y_{uis}(\mathbf{x}_u * \mathbf{z}_i)^T + 2\sum_{u,i} c_{uis}(\mathbf{x}_u^2 * \mathbf{z}_i^2)^T * \mathbf{v}_s^T + 2\lambda \mathbf{v}_s^T, \tag{3.16}$$

for all $v \in \mathscr{U}$, $j \in \mathscr{I}$, $s \in \mathscr{T}$. Setting the first order derivatives (3.14) - (3.16) equal to zero, and solving for the free variable, we get the three first order conditions we need to solve with ALS. Pseudocode for solving the $f(U+I+T)$ feature entries using ALS is given in Algorithm 5. However this implementation is computationally slow and Hidasi & Tikk (2012) manipulates the expressions for a significant speedup. They do this over several pages and Algorithm 5 is thus just the slow simple one.

Algorithm 5 Simple alternating least squares for WRCP

procedure ALSWRMF($\underline{\mathbf{C}}, \underline{\mathbf{Y}}, f, \lambda, E$)
  initialize:
  $\mathbf{X} \in \mathbb{R}^{U \times f}$, $\mathbf{Z} \in \mathbb{R}^{I \times f}$, $\mathbf{V} \in \mathbb{R}^{I \times f}$ with small random numbers
  for epoch $= 1, \ldots, E$ **do**
    for $u = 1, \ldots, U$ **do**
      $\mathbf{x}_u = \left(\sum_{i,t} c_{uit}(\mathbf{z}_i^2 * \mathbf{v}_t^2) + \lambda \mathbf{1}\right)^{-1} * \sum_{i,t} c_{uit}\, y_{uit}(\mathbf{z}_i * \mathbf{v}_t)$
    for $i = 1, \ldots, I$ **do**
      $\mathbf{z}_i = \left(\sum_{u,t} c_{uit}(\mathbf{x}_u^2 * \mathbf{v}_t^2) + \lambda \mathbf{1}\right)^{-1} * \sum_{u,t} c_{uit}\, y_{uit}(\mathbf{x}_u * \mathbf{v}_t)$
    for $t = 1, \ldots, T$ **do**
      $\mathbf{v}_t = \left(\sum_{u,i} c_{uit}(\mathbf{x}_u^2 * \mathbf{z}_i^2) + \lambda \mathbf{1}\right)^{-1} * \sum_{u,i} c_{uit}\, y_{uit}(\mathbf{x}_u * \mathbf{z}_i)$
  return $\mathbf{X}, \mathbf{Z}, \mathbf{V}$

Algorithm for finding optimal values of $\mathbf{X}$, $\mathbf{Z}$ and $\mathbf{V}$ and in the Weighted regularized matrix factorization model.

3.3.2 Weighted Regularized Tucker Decomposition

The other tensor decomposition considered in this thesis is the Tucker decomposition where a tensor is decomposed into three matrices and a core tensor. This is not much covered in the CF literature, quite likely because of the computational complexity, and the following adaption from CP to the Tucker

decomposition is the author's work. The interaction of the matrices with the core, makes the decomposition able to handle users' fluctuations in taste more individually across contexts. We model user u's preference for item i under context t as three latent factor vectors, one for each of u, i and t, and multiplied together with each mode of the core tensor. We now allow the entities to have different numbers of «features», we do not map them to the same latent space anymore, nonetheless we will still call it latent features. The score for user u, item i with context t now is

$$\hat{y}_{uit} = \sum_{a,b,c}^{d_x,d_Z,d_V} s_{abc} x_{ua} z_{ib} v_{tc}. \tag{3.17}$$

Recall that the Tucker decomposition consists of three feature matrices and one core tensor. The core tensor captures the interaction of the three entities *USERS*×*ITEMS*×*CONTEXT*. The loss function with Tucker decomposition becomes

$$L(\mathbf{X},\mathbf{Z},\mathbf{V},\underline{\mathbf{S}}) = \sum_{u,i,t} c_{uit} \left(y_{uit} - \sum_{a,b,c}^{d_X,d_Y,d_Z} s_{abc} x_{ua} z_{ib} v_{tc} \right)^2 + \lambda \left(\sum_u \|\mathbf{x}_u\|_F^2 + \sum_i \|\mathbf{z}_i\|_F^2 + \sum_t \|\mathbf{v}_t\|_F^2 \right).$$

In this model we have a quite sizable $Ud_X + Id_Z + Td_V + d_X d_Z d_V$ number of feature entries to estimate and the two tuning parameters α and λ to choose. Herein lies one of the biggest drawbacks, the complexity of the model is exponential in the number of context variables and polynomial in the size of the factorization. From (2.11) we know that we can rewrite the problem as

$$\min_{\mathbf{X},\mathbf{Z},\mathbf{V},\mathbf{S}_{(1)}} \left\{ \left\| \mathbf{C}_{(1)}^{1/2} * \left(\mathbf{Y}_{(1)} - \mathbf{X}\mathbf{S}_{(1)}(\mathbf{V}\otimes\mathbf{Z})^T \right) \right\|_F^2 + \frac{\lambda}{2} \left(\|\mathbf{X}\|_F^2 + \|\mathbf{Z}\|_F^2 + \|\mathbf{V}\|_F^2 \right) \right\}.$$

This way of writing makes it relatively easy to find the equation for $\mathbf{x}_u$ in the ALS algorithm. If we keep all other unknown constant we can find the minimization for user $\mathbf{x}_u$ by minimizing

$$f(\mathbf{x}_u) = \left(\mathbf{y}_{(1),u} - \mathbf{x}_u^T \mathbf{S}_{(1)}(\mathbf{V}\otimes\mathbf{Z})^T \right) \mathbf{C}_u \left(\mathbf{y}_{(1),u} - \mathbf{x}_u^T \mathbf{S}_{(1)}(\mathbf{V}\otimes\mathbf{Z})^T \right)^T + \frac{\lambda}{2} \mathbf{x}_u^T \mathbf{x}_u,$$

where $\mathbf{y}_{(1),u}$ is row u in the matrix $\mathbf{Y}_{(1)}$, $\mathbf{x}_u$ is the row of $\mathbf{X}$ as a column vector and $\mathbf{C}_u$ is a diagonal matrix with the entries of row u of $\mathbf{S}_{(1)}$ on the diagonal.
and likewise for z and v. To update the x in the ALS algorithm we need to solve the first order condition

$$\left(\mathbf{S}_{(1)}(\mathbf{V}\otimes\mathbf{Z})^T \mathbf{C}_u (\mathbf{V}\otimes\mathbf{Z}) \mathbf{S}_{(1)}^T + \lambda \mathbf{I} \right) \mathbf{x}_u = \mathbf{S}_{(1)}(\mathbf{V}\otimes\mathbf{Z})^T \mathbf{C}_u \mathbf{y}_{(1),u}^T,$$

for $\mathbf{x}_u$. The solutions for $\mathbf{z}$ and $\mathbf{v}$ are

$$\mathbf{z}_i = \left(\mathbf{S}_{(2)}(\mathbf{V}\otimes\mathbf{X})^T\mathbf{C}_i(\mathbf{V}\otimes\mathbf{X})\mathbf{S}_{(2)}^T + \lambda\mathbf{I}\right)^{-1}\mathbf{S}_{(2)}(\mathbf{V}\otimes\mathbf{X})^T\mathbf{C}_i\mathbf{y}_{(2),i}^T$$
$$\mathbf{v}_t = \left(\mathbf{S}_{(3)}(\mathbf{Z}\otimes\mathbf{X})^T\mathbf{C}_t(\mathbf{Z}\otimes\mathbf{X})\mathbf{S}_{(3)}^T + \lambda\mathbf{I}\right)^{-1}\mathbf{S}_{(3)}(\mathbf{Z}\otimes\mathbf{X})^T\mathbf{C}_t\mathbf{y}_{(3),t}^T,$$

and can be found in the same way by using the two remaining modes of unfolding. Kolda & Bader (2009) show that the core tensor $\underline{\mathbf{S}}$ must satisfy

$$\underline{\mathbf{S}} = \underline{\mathbf{Y}} \times_1 \mathbf{X}^T \times_2 \mathbf{Z}^T \times_3 \mathbf{V}^T.$$

The ALS Algorithm 6 shows pseudocode for how we can solve the weighted regularized Tucker decomposition model. We see from the closed form solutions for $\mathbf{x}_u$, $\mathbf{z}_i$ and $\mathbf{v}_t$ that terms such as $\mathbf{S}_{(1)}(\mathbf{V}\otimes\mathbf{Z})$ for $\mathbf{x}_u$, are included in the updates of every user, item and context category. Algorithm 6 would be significantly faster if we utilized these easy speedups and did these computations outside the inner loops.

Algorithm 6 Simple alternating least squares for Weighted regularized Tucker decomposition

procedure ALSTUCKER($\underline{\mathbf{C}}, \underline{\mathbf{Y}}, d_X, d_Z, d_V, \lambda, E$)
  initialize:
  $\mathbf{X}\in\mathbb{R}^{U\times d_X}$, $\mathbf{Z}\in\mathbb{R}^{I\times d_Z}$, $\mathbf{V}\in\mathbb{R}^{I\times d_V}$, $\underline{\mathbf{S}}\in\mathbb{R}^{d_X\times d_Z\times d_V}$ with small random numbers
  for epoch $= 1,\ldots,E$ **do**
    for $u = 1,\ldots,U$ **do**
      $\mathbf{x}_u = \left(\mathbf{S}_{(1)}(\mathbf{V}\otimes\mathbf{Z})^T\mathbf{C}_u(\mathbf{V}\otimes\mathbf{Z})\mathbf{S}_{(1)}^T + \lambda\mathbf{I}\right)^{-1}\mathbf{S}_{(1)}(\mathbf{V}\otimes\mathbf{Z})^T\mathbf{C}_u\mathbf{y}_{(1),u}^T$
    for $i = 1,\ldots,I$ **do**
      $\mathbf{z}_i = \left(\mathbf{S}_{(2)}(\mathbf{V}\otimes\mathbf{X})^T\mathbf{C}_i(\mathbf{V}\otimes\mathbf{X})\mathbf{S}_{(2)}^T + \lambda\mathbf{I}\right)^{-1}\mathbf{S}_{(2)}(\mathbf{V}\otimes\mathbf{X})^T\mathbf{C}_i\mathbf{y}_{(2),i}^T$
    for $t = 1,\ldots,T$ **do**
      $\mathbf{v}_t = \left(\mathbf{S}_{(3)}(\mathbf{Z}\otimes\mathbf{X})^T\mathbf{C}_t(\mathbf{Z}\otimes\mathbf{X})\mathbf{S}_{(3)}^T + \lambda\mathbf{I}\right)^{-1}\mathbf{S}_{(3)}(\mathbf{Z}\otimes\mathbf{X})^T\mathbf{C}_t\mathbf{y}_{(3),t}^T$
    $\underline{\mathbf{S}} = \underline{\mathbf{Y}} \times_1 \mathbf{X}^T \times_2 \mathbf{Z}^T \times_3 \mathbf{V}^T$
  return $\mathbf{X}, \mathbf{Z}, \mathbf{V}, \underline{\mathbf{S}}$

Algorithm for finding optimal values of $\mathbf{X}$, $\mathbf{Z}$ and $\mathbf{V}$ and $\underline{\mathbf{S}}$ in the Weighted regularized Tucker decomposition model.

Chapter 4

Data and Evaluation Methodology

In this chapter I start by describing the NRK TV dataset containing implicit feedback, in Section 4.1. Later, in Section 4.2 we select a subset of the data on which to test the collaborative filtering methods of Chapter 3 on. Section 4.3 introduces the evaluation methodology we will use to compare how well the methods perform.

4.1 Dataset

The NRK TV dataset consists of 5,705,400 observations from the end of January 2016 to the end of November the same year. Each observation has nine covariates, and the data is collected from NRK's online television platform NRK TV. The data does not include programs from NRK's television platform *NRK Super*, which is intended for kids.

The nine covariates are *cookie ID*, *program ID*, *content ID*, *content type*, *action*, *visit start time*, *time offset*, *device category* and *app ID*. The *cookie ID* is a random string given by Google Analytics[1] to be able to identify that different hits belong to the same user. The cookie ID is what defines users in this text, although a real person/user may have several devices and thus, several cookie IDs. The cookie IDs have been anonymized by NRK. *Program ID* is a unique string for each program, or episode of a series of which there are 41,743 in the dataset. *Content ID* is the name of the program or series. There are 7,121 different content IDs so there are quite a few series. The *content type* can be either series or program, which indicates if the program is part of a series or not. *Action* is a bit more complicated. Depending on which device and which app it can either be registered as started, halfway or completed, or started, 25 %, halfway, 75 % or completed. *Visit start time* is given as a UNIX timestamp i.e. seconds since

[1] `analytics.google.com/`

Table 4.1: Quantiles of events per user and program

Quantile	0 %	25 %	50 %	75 %	100 %
Program events	1	2	5	26	642,343
User events	1	2	5	15	1,934

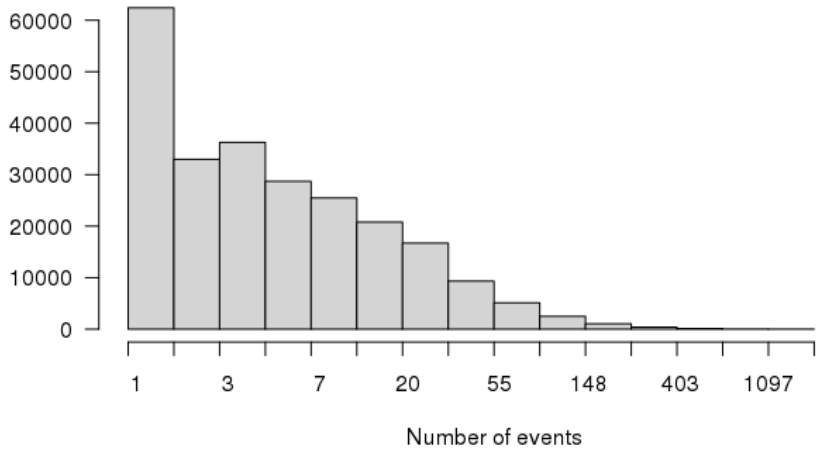
January 1, 1970. *Time offset* is given in milliseconds from visit start time. *Device category* shows if the event occurred on a desktop, a mobile phone or tablet. The last one is *App ID*, which shows if the user watched on the iPhone/iPad-app, Android-app or on NRK's web page.

The dataset described above, is complemented by a meta dataset with information about all the content the users have interacted with. It tells the duration of the program, date of first transmission/publication, year of production, category, legal age and episode number.

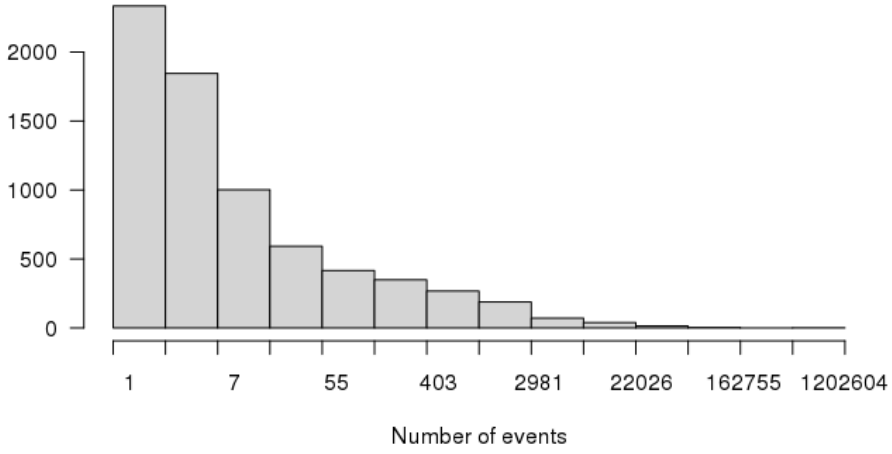
There are 184,268 distinct users and 7,121 series and programs. If we consider each individual episode by itself, the dataset consists of 41,743 different episodes and programs. By removing duplicates of the duplets (user ID, program ID) only 2,206,515 of the observations remain. This gives a U-I matrix with a sparseness of 0.9983.[2] Many of the observations in the dataset are for the same user, same item in same session. The system has then registered each value of the view variable has taken. We are only interested in the last of these observations, how much the user ended up seeing of the program. If we allow for rewatching a program at another start time, thus removing duplicates of the triplets (user ID, program ID, start time) from the dataset, we end up with 2,886,535 observations. These observations will be referred to as events. An event thus shows how much of a program the user watched in a session. There is a possibility that one would rewatch a program in the same session, but this is disregarded in the following. We can see from Figure 4.1a and Table 4.1 that the distribution of number of events among users is heavily right-tailed. Note the use of log scale on the x-axis in the histograms in Figure 4.1. For programs the story is even more extreme as can be seen from Figure 4.1b and Table 4.1. The outlier for programs is the Norwegian hit show Skam, that was very popular in the time period the dataset is from. It alone is responsible for 22 % of all events. We can see how it trumps the other programs in Figure 4.2. The figure shows the number of events for the five most popular programs in the dataset, and it is clear that Skam has been very popular.

I have given the actions started, 25 %, halfway, 75 % and completed the numerical values 0.10, 0.25, 0.50, 0.75 and 1.00, respectively. Figure 4.3 shows the frequencies for the actions in the dataset. Almost 59 % of started programs get

[2] $1-2206515/(184268*7121)$



(a) Users



(b) Programs

Figure 4.1: Histograms of number of events for users and programs in log scale. Both histograms are heavily right-tailed.

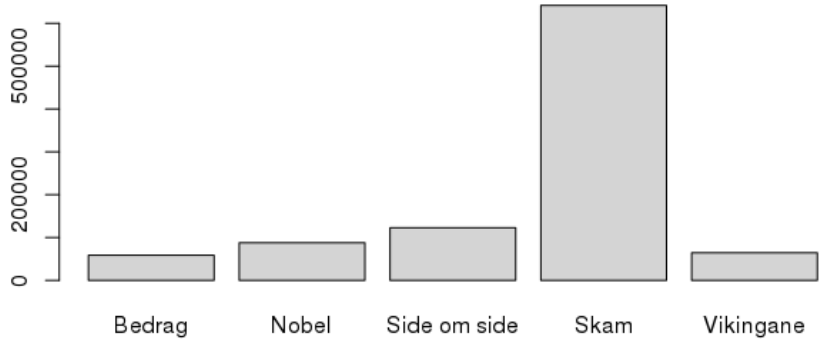


Figure 4.2: The number of events for the five most popular series in the dataset. Skam has almost twice as many events as the other four put together.

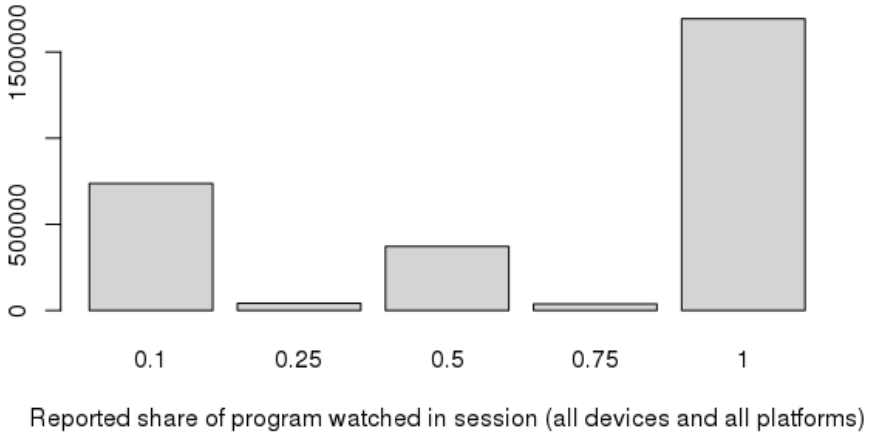


Figure 4.3: Plot of the share of program watched in session.

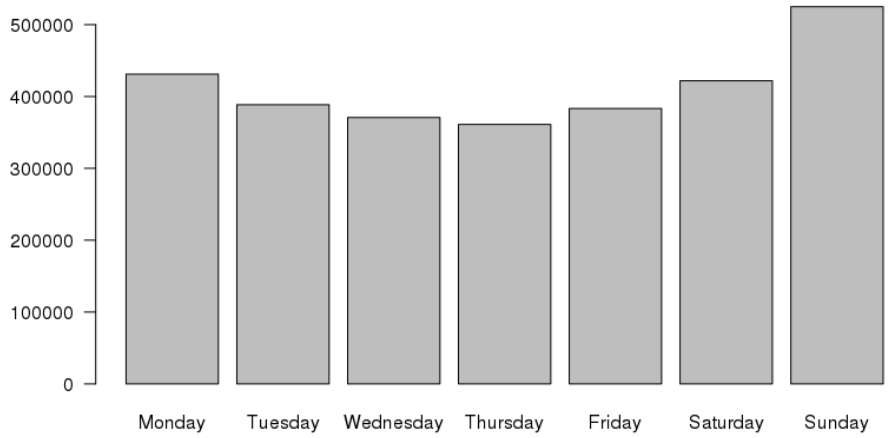


Figure 4.4: Barplot of events and weekdays.

finished in the same session. We can see that 25 % and 75 % are much less common. This is partly because these actions are only registered if the app ID is *web*, but more significantly because the system NRK uses only started registering these actions in the middle of the period in which the data was collected. Figure B.1 in the appendix indicates that fewer users watch 25 % and 75 % of a program on mobile and tablet devices compared to desktop. This is likely because mobile and tablet users use the NRK app. Another trait we can note from Figure B.1 is that we have a higher percentage of *started* on mobile. Maybe it it harder to commit to programs on phones, or maybe it is easier to find another interesting program on mobile, so one briefly tests programs to get a feel for them.

One factor of interest is when users are using the service. The barplot in Figure 4.4 shows that Sunday is the most popular day for streaming. It looks like the usage has a top on Sunday and then goes down gradually in both «directions», with Thursday as the least popular day. We are also interested in when users are watching during the day. The histogram in Figure 4.5 shows when users finish or stop watching a program, e.g. the time of events. We see that the use of the service is quite low and stable during office hours, then it increases gradually until close to midnight. This behavior seems to be pretty similar for all devices and app IDs, which can be seen from Figure B.3 and Figure B.4 in the appendix. If we compare when users are watching during the weekdays Monday - Friday, compared to Saturday - Sunday, we can see from Figures B.5a and B.5b that more are using the service in the office hours 9-16, in the weekend.

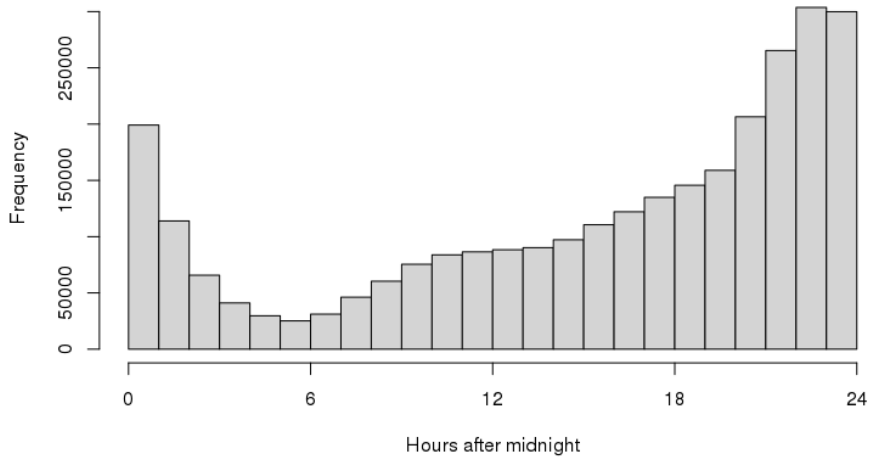


Figure 4.5: Histogram of when events occur during the day. An event is when a user finishes or stops watching a program.

4.2 Selecting Data to Work With and Creating Training Sets

It is a difficult task to give good, even decent, personalized recommendations to users with very few events. Because of this, and because of the fact that the algorithms are pretty time consuming, I chose to remove about half of the events by removing events for users and programs/series with less than 20 events. Note that all of the user's 20 events could be with 20 different episodes of the same program. It could also be re-watching the same episode of a series 20 or more times. And likewise, all events for a series could be from one fan. By removing these events we are left with a subset of 1,940 unique content IDs and 35,276 NRK cookie IDs. This results in a U-I matrix of 69,312,320 elements opposed to 1,312,172,428, but it is still a quite respectable size. The level of sparseness is reduced even more. The U-I matrix now has a level of sparseness of 0.9926. The implicit ratings in the U-I matrix, **R**, are created by summing together all events of a program/series the user has watched. So if a user has watched one and a half episodes of a series, this entry would have a value of 1.5 in **R**. I have made the decision to divide the columns for series by a factor of 2. This is because episodes of series often are shorter than «independent» programs shown on NRK. The distribution of the values of non-zero elements in **R** is very heavy tailed with more than half of the entries less than one, and the highest value is 456.

4.3 Evaluation Methodology

4.3.1 What Makes a Good Recommender System?

Before we start to evaluate and compare the different models in Chapter 3, we need to know what we want out of a RS. It might seem obvious what distinguishes a good RS from a bad one. A good RS should recommend items that the user finds relevant. The more interesting the item is to the user, the better it is. The service provider's primary interest is, most often, to earn money, and to him the best possible RS is the one that would make the most profit. The perspective of profit does not necessarily rule out the users' needs, quite the opposite. User satisfaction is extremely important to the service provider.

Three central concepts for a RS, that go beyond making relevant recommendations, are *novelty*, *serendipity* and *diversity*. A novel recommendation is a recommendation of an item the user does not know about. This information is very hard to get. From the events we know some of the items the user knows about, but they seldom interact with all items they know about in the catalog. Serendipity tells about how surprising a successful recommendation was. If the user enjoys the works of a certain director and gets recommended a movie that he does not know about from this director, we would consider this a novel recommendation, but not surprising as it would be a safe bet that the user would find the movie interesting. Recommending the item with least events would be very surprising, but it might be unpopular for a reason, and the user might find it uninteresting. Some services, like NRK, also want to give diverse recommendations. They value that a big part of their catalog gets recommended.

There is however, no doubt that the most important objective is to recommend items the user finds interesting. We will look at metrics to evaluate how well the methods perform in the following section.

4.3.2 Evaluation Metrics

When evaluating a RS we are most interested in how good its future recommendations will be. The gold standard method of evaluating RS is to do online A/B (All Between) testing , i.e. for group A use the RS to give recommendations and for group B don't use the RS to give recommendations (or use another RS), and then compare the results. With A/B testing we can also look at long term effects as the user gets accustomed to the RS. However, this evaluation method is usually only possible for the service provider, as it has to be done online, and thus unfortunately not used in academia where the usual case is offline experiments. Offline experiments are performed with pre-collected, static, data with user interactions (Shani & Gunawardana 2011), as is the case with the NRK TV dataset. For collaborative filtering models with explicit feedback the dataset includes the users' ratings and one measures the difference in predicted rating

and actual rating for a test set. Let **R** denote the rating matrix. The element of **R**, r_{ui}, is then the rating if user u has rated item i, or else an empty entry. We then divide the ratings into a training set and a test set. In the training set a share of the ratings will be set to empty entries and we will use these to compare how well the system did in the test set. For this section we will call the test set $\mathscr{T}$ and it will consist of the entries in **R** set to unknown. The most used measure for how good the RS performs, especially for prediction problems, is the root mean squared error

$$\text{RMSE} = \sqrt{\frac{1}{|\mathscr{T}|} \sum_{(u,i)\in\mathscr{T}} (r_{ui} - \hat{r}_{ui})^2},$$

where user u's predicted rating for item i is $\hat{r}_{ui}$. RMSE was the measure used in the Netflix-prize, and the goal was to improve the RMSE from Netflix' own algorithm by 10 % on an independent test set.

Another popular measure for prediction problems is the mean absolute error

$$\text{MAE} = \frac{1}{|\mathscr{T}|} \sum_{(u,i)\in\mathscr{T}} |r_{ui} - \hat{r}_{ui}|.$$

The MAE is often preferred because it is less sensitive to outliers compared to RMSE. However, how do we evaluate the recommendations if we do not know how well the user likes any of the items?

In most situations the goal of recommender systems is to make the best possible list of recommended items, a top-N list. The measures described above are used for prediction accuracy and not for ranking or classification. When we work with implicit feedback we do not know how much the user like the items, but an interaction is a positive signal. A common way to evaluate a top-N list is to see how many of the N recommended items the user has interacted with, given that they were adjusted to not-interacted with in the training data.

4.3.3 Receiver Operating Characteristics

So how can we evaluate the list of N recommended items? One solution is to classify the recommendations as (i) true positive (tp), i.e. the recommendation is interesting to the user, (ii) a true negative (tn), i.e. a not interesting item is not recommended to the user, (iii) a false positive (fp), i.e. a recommended item is not interesting to the user, and (iv) a false negative (fn), i.e. the user is not recommended an interesting item. Here interesting to the user will be if they have interacted with the item. The confusion matrix in Table 4.2 consists of these four possibilities. The upper row will be the total number of recommended items, number of users times N. The left column has the total number of

Table 4.2: Confusion Matrix.

	Actual positive (in test set)	Actual negative (not in test set)
Recommended	tp	fp
Not recommended	fn	tn

possible correct predictions. Ideally we would have low false positives and false negatives and high of the two others.

Two popular popular measures of evaluation that utilizes the confusion matrix, is *precision* and *recall* defined below.

$$precision = \frac{\text{tp}}{\text{tp}+\text{fp}} \tag{4.1}$$

$$recall = true\ positive\ rate = \frac{\text{tp}}{\text{tp}+\text{fn}} \tag{4.2}$$

$$false\ positive\ rate = \frac{\text{fp}}{\text{fp}+\text{tn}} \tag{4.3}$$

The precision is defined as the ratio of true positive recommendations to the total number of recommended items, true positives and false positives or the upper row in the confusion matrix. Recall is the ratio of true positive recommendations to the number of interesting items (true positives and false negatives) and is also called the hit rate. Optimally we would have both precision and recall equal to one, but there is a trade off here. We could easily recommend every item and achieve recall equal to one, alas this would lead to a high number of false positives and thus a low precision.

Precision and recall is widely used, but most like to visualize these results through the closely connected receiver operating characteristics (ROC) curves (Davis & Goadrich 2006). ROC curves plot the *true positive rate* (recall) against the *false positive rate* for different values of N. Perfect performance will equate to an area under the curve (AUC) equal to 1, and if it is totally random the AUC will be 0.5. In Chapter 5 we will use partial ROC curves to evaluate which algorithms that performs best. The reason for using partial ROC curves is that the case where the true positive rate and the false positive rate are both close to 1 does not interest us much. It will be for a high number N such that the top-N list would not be easily presented to the user anyway.

4.3.4 Dataset Partitioning and Application

The model validation technique used in this thesis will be what is known in the literature as *holdout*. With holdout we split the data into two parts, a training set and a test set. This is done for 5 folds so that each of the test sets have

no common elements. In each test set a share of the users will have the 5 programs/series they have interacted with most recently removed. The goal is then to recommend the programs removed from the training set to the user, and we will call this successful prediction. We choose the model's parameters by recall-12, i.e. for each fold we train the model on the training set, then recommend 12 items to each user in the test set and count the number of successful predictions. The number of successful predictions are then summed together across the folds, and the parameter combination that achieves the highest number of true positives is chosen. This is equivalent to recall since the denominator in (4.2) is constant. Because we choose the best combination of all the combinations we try, this recall-12 result tends to exaggerate the models performance. Therefore we will visualize how well it does with this parameter combination, for different values of N in the top-N list, using the ROC curve. We will also use these ROC curves to compare different models' performance against each other.

I chose holdout over the more traditional cross validation as validation method, because it allows for more control when splitting the data into training sets and test sets. Traditional k-fold cross validation splits the data into k independent sets and uses each set as a test set one time. Collaborative filtering techniques like matrix factorization needs a handful of observations to be able to catch the users' latent features Cremonesi et al. (2008), and I chose to only remove events for users with more than 20 events. This selection could lead to a bias as it is not random, but in which way is not obvious.

4.3.5 If Recommendations Were Random

What would be the expected number of successes if we just recommended a random list of N programs to each user? First of all we would have a terrible RS! So this section is just meant to set the bar very low for later results. In the dataset discussed earlier, we removed 5 programs for 6213 users to serve as test observations. If X_i denotes the number of successful guesses for user i in the test set, X_i follows a hypergeometric distribution with probability mass function

$$p(X = x) = \frac{\binom{5}{x}\binom{1940-W+5}{N-x}}{\binom{1940-W+5}{N}},$$

for $x = 0, 1, 2, 3, 4, 5$ where W is the number of different programs/series the user has watched/interacted with. We do not recommend items the user already has interacted with. The expected value of correctly randomly guessed recommendations is the sum of expected values, and equals 195 when $N = 12$. This is very low and very small adjustments would lead to a much better RS.

Chapter 5

Results

In this chapter I apply the models from Chapter 3 to suitable NRK TV data. The first part focuses on algorithms on the simple dataset discussed in the previous chapter. In Section 5.2 we try to find good context categories, and then test to see how well the algorithms perform with context in Section 5.3. A modified version of WRMF, that arguably performs better, is proposed in Section 5.4.

5.1 Results Without Context

This section will focus on traditional latent features CF with matrix factorization, i.e. we only consider the U-I matrix $\mathbf{R}$. The U-I matrix consists of 35,276 rows (users) and 1,940 columns (items).

5.1.1 Weighted Regularized Matrix Factorization

WRMF was introduced in Section 3.2.3, and it will form the baseline when we compare the models. The objective of WRMF, as with most MF, is to find the two matrices with users' and items' latent features, that multiplied best predict how much users will enjoy the programs. More specifically we want to find

$$\mathbf{X}^*, \mathbf{Z}^* = \underset{\mathbf{X}\in\mathbb{R}^{U\times f}, \mathbf{Z}\in\mathbb{R}^{I\times f}}{\arg\min} \sum_{u,i} (1+\alpha r_{ui})(y_{ui} - \mathbf{x}_u^T \mathbf{z}_i)^2 + \lambda \left(\|\mathbf{X}\|_F^2 + \|\mathbf{Z}\|_F^2 \right), \qquad (5.1)$$

where the confidence function $c = 1 + \alpha r$, is included for clarity. With f latent factors we need to estimate $(35,276 + 1,940)f$ feature entries, and from (5.1) we see that there are two tuning parameters to choose as well, the regularization parameter λ and the confidence parameter α. Choosing the parameters for the WRMF model from Section 3.2.3 is done by 5-fold holdout described in Secion 4.3.4. In each fold 1,243 unique users with more than 20 events have 5 of the programs they have interacted with set to zero in $\mathbf{R}$, and thus $\mathbf{Y}$, for testing. After finding the best parameter combination, the model is evaluated by its partial ROC curve.

There are by construction an infinite number of parameter combinations of α and λ to choose from. In this section 576 parameter combinations have been tried. The values tested are chosen because they are popular in literature and practice, and they performed well during initial testing. There is however, definitely room for more fine tuning. The values for f, the number of factors, in the validation were 1, 2, 3, 4, 5, 8, 10, 12, 15, 20, 25 and 30. Tested values for the regularization parameter λ were 0.005, 0.01, 0.015, 0.025, 0.05 and 0.1. A common starting point for α is to choose it such that the positive (non-zero) and negative (zero) entries have roughly the same weight (B. Frederickson, personal communication, March 1, 2017). The starting point, α_0 solves

$$\sum_{u,i:r_{ui}=0} 1 = \sum_{u,i:r_{ui}>0} (1+\alpha_0 r_{ui}),$$

where the left hand side is the number of negative entries and the right is the total confidence in the positive entries of **R**. With the U-I matrix this gives $\alpha_0 = 96$ as starting point. This proved to be way too high for the data, and the results presented in this section have α values 1, 2, 5, 7, 10, 15, 20, 30 which give higher numbers of successful predictions. Altogether, this gives the 576 combinations. The implementation of the algorithm is from the **implicit** Python module (Frederickson 2017). In the 5-fold holdout each user is recommended a list of $N = 12$ items that he has not interacted with in the training data. The number of iterations, epochs in Algorithm 3, were set to 50. This is considered to be quite high and should thus ensure good convergence.

The number of successful recommendations for the different values of factors and regularization is visualized in Figure 5.1. The tiles in the heatmap shows the best result obtained with the combination of f and λ, i.e. the combination of f, λ and α with best result is shown. The figure clearly shows that $f = 10$ gives best results and that regularization does not seem to have a very big effect. The best result of 9,701 successful predictions is achieved with $f = 10$, $\lambda = 0.01$ and the «hidden» $\alpha = 5$, not shown in the figure. 9,701 out of 31,065 possible is magnitudes better than the expected 195 successes from random guessing, and gives a success rate of 31.2 %. It is difficult interpret how good this percentage is. If the service has a very big catalog, the user might not know about the items he would enjoy the most. The WRMF results used for comparison in the next Section 5.1.2) are computed with the best parameter combination $f = 10, \lambda = 0.01, \alpha = 5$.

When we have computed the item feature matrix **Z**, it is easy to find out which programs that have similar features. We could of course, do this with **X** to find similarly minded users as well, but this is more difficult for the user to relate to. A small selection of six programs are compared using the cosine similarity, in Figure 5.2. The figure shows the cosine similarities, defined in (3.1). *Over hekken* and *Side om side* have a cosine similarity of 0.99, which suggests

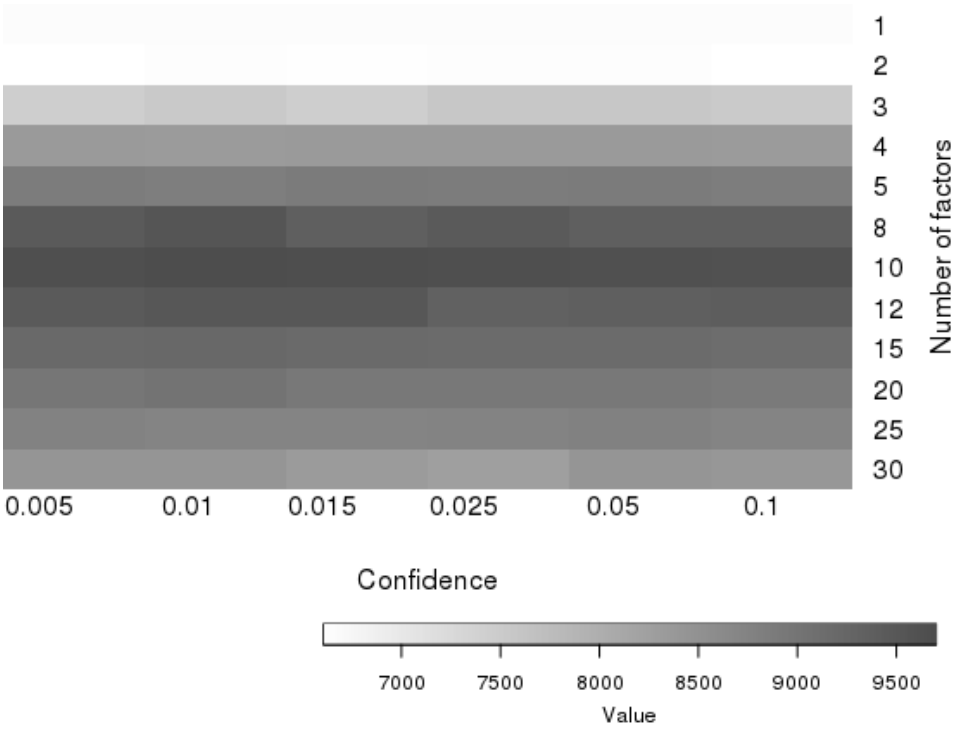


Figure 5.1: Heatmap of successful predictions for WRMF with different combinations of number of factors (f) and regularization parameter (λ). The confidence parameter α is the one that gave the highest number of successes for each tile. Value is the number of successful predictions.

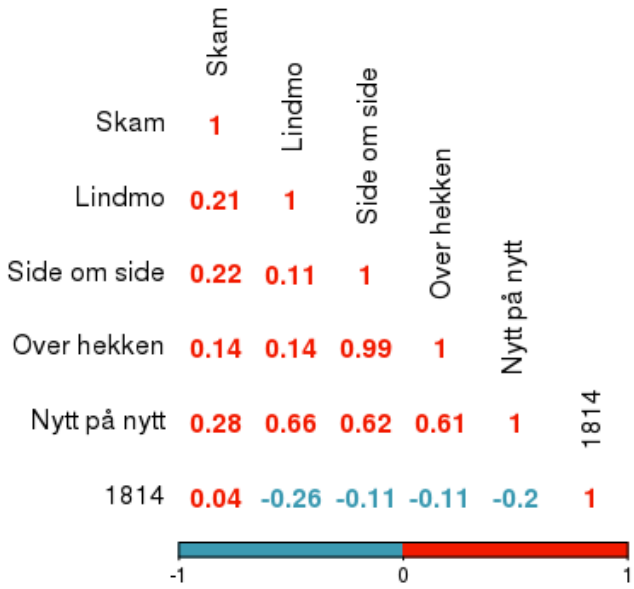


Figure 5.2: Cosine similarities of six programs' feature vectors learned by WRMF with $f = 10$ and $\lambda = 0.005$.

that they are very similar. The first is actually a spin off of the latter so this is reasonable. A user who has watched only one of them is very likely to get the other one recommended. The historical mini-series *1814* is, judging by the cosine similarities, quite different from all the other programs. Users who get recommended *1814* will not be likely to get any of the other five in their to-N list. Latent factor models makes it easy to find similar items (and users) and this could be used to recommend similar programs to those the user has already interacted with. Being able to explain why the item is recommended, is often seen as a very favorable property for a RS. This is not to imply that this is exactly what happens with WRMF, where the preference scores are made from the inner product of the user's and item's feature vector. Nonetheless, users will tend to get recommendations of items that have similar features as the ones they have interacted with, this is also true for the cosine similarity. In the following section we look at the Bayesian Personalized Ranking Criterion with matrix factorization and compare it with WRMF.

5.1.2 Bayesian Personalized Ranking With Matrix Factorization

To use the BPR with MF introduced in Section 3.2.4, we need to define the set

$$\mathscr{D} = \left\{(u,i,j) : u \in \mathscr{U},\ i \in \mathscr{I}_u^+,\ j \in \mathscr{I} \setminus \mathscr{I}_u^+\right\}, \tag{5.2}$$

with triplets of users that prefer item i to j. Here we do not take into consideration how many times the user has interacted with the item i, we disregard information. Recall the function we want to maximize

$$\sum_{(u,i,j)\in\mathscr{D}} \left\{ \ln \frac{1}{1+e^{-\hat{y}_{uij}}} - \frac{\lambda}{2}\left(\mathbf{x}_u^T\mathbf{x}_u + \mathbf{z}_i^T\mathbf{z}_i + \mathbf{z}_j^T\mathbf{z}_j\right)\right\}.$$

The method has one less parameter to estimate as we, for better and worse, do not have the confidence parameter.[1] BPR with MF is another standard algorithm found in several publicly available libraries. The implementation used here is from the R package **rrecsys** (Çoba 2016). The values used for f and λ in the evaluation were 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 75, 85, 100, 125, 150, 175, 200 and 0.005, 0.01, 0.015,0.025, 0.05, 0.1 respectively. This gives the 120 tiles in Figure 5.3. We note that the results do not change as gradually as with WRMF. The best results are quite far away from each other and often a small change in either number of factors or regularization changes the result dramatically. The dark tile, (0.1, 150), has quite white tiles for both neighboring number of factors. This result is also found in Table A.1, where we see that factors vary a lot for the 10 best combinations, especially compared to WRMF where $f = 10$ dominated. Both Figure 5.3 and Table A.1 indicates that BPR with MF performs significantly worse than WRMF with $N = 12$ recommendations per user. To get the bigger picture of which algorithm performs best on the data we want to see how well they perform with different number of N in the top-N list recommended to users. We achieve the highest number of successes, 7,335, with $f = 50$, $\lambda = 0.025$. However the rest of the top 10 highest number of successful predictions spans the big interval from 5 to 150 factors. The second best result with 7,334 successes was produced with $f = 5$ and a higher regularization value of 0.1. Still, the winner is $f = 50$ and $\lambda = 0.025$, and the ROC curves are computed with this combination.

The partial ROC curves in Figure 5.4 are very favorable for WRMF. It is better across the board and the precision vs recall plot in Figure 5.5 tells the same story. WRMF performs better on the dataset. Compared to WRMF, the recall for BPR with MF is very linear for N higher than 5. It is in this range it seems like its performance is lowest.

Another metric that is relevant to NRK is diversity. The cumulative share of recommended programs vs the cumulative share of recommendations for WRMF is shown in Figure 5.6a and for BPR with MF in Figure 5.6b. These plots are across all users and all folds, so all 35,276 users are recommended 20 items five times. Ideally, if we want high diversity, this line should be straight. While they both are far from straight, we can see that WRMF does better at diversity as the area under the curve is bigger, plus it also recommends more items in total.

[1] This assumes that we use the same regularization parameter for users, the preferred item, i in $\mathscr{D}$, and the inferior item j.

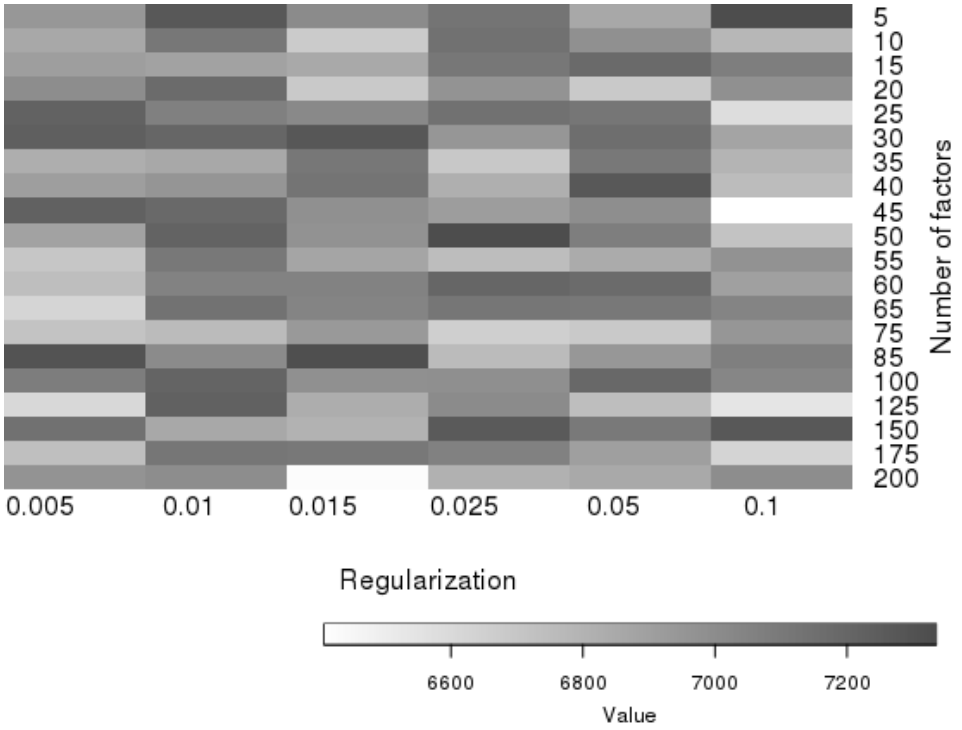


Figure 5.3: Heatmap showing the number of successful predictions for Bayesian personalized ranking with matrix factorization with different numbers of factors and regularization parameter. We see that the darkest tiles with the highest values are quite far apart.

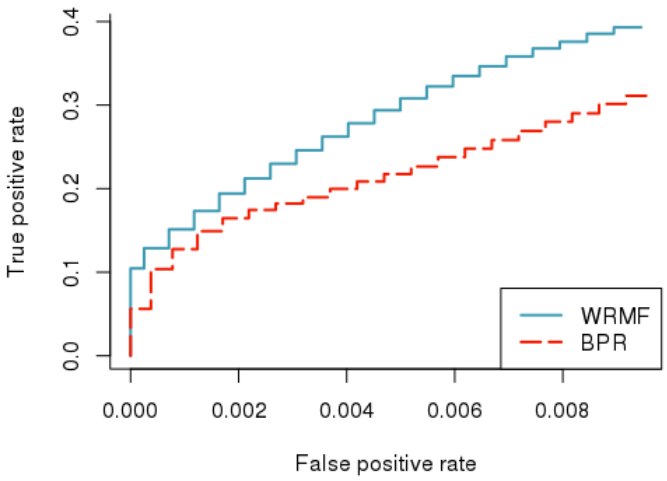


Figure 5.4: Partial ROC curves for WRMF and BPR with MF for $N = 0, 1, ..., 20$.

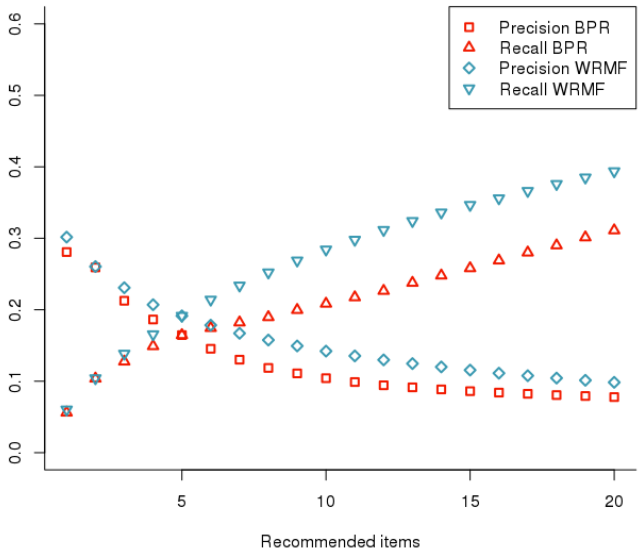
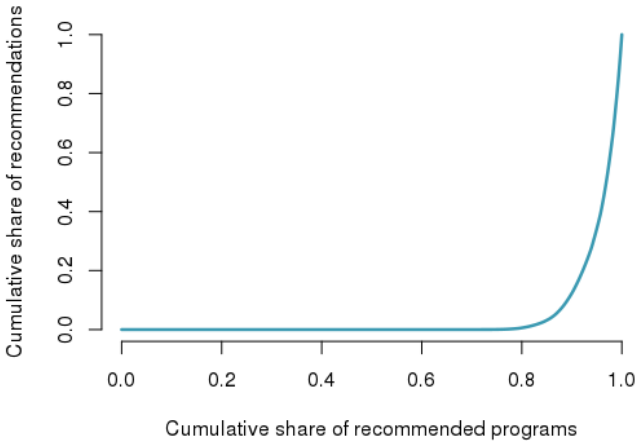
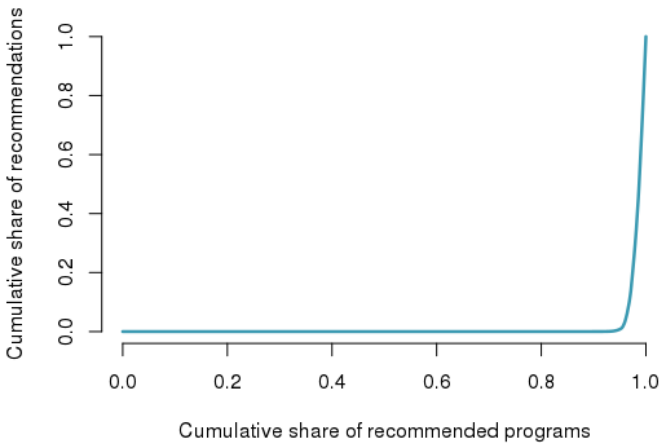


Figure 5.5: Precision vs recall for WRMF and BPR with MF for $N = 0, 1, ..., 20$.



(a) WRMF on folds 1-5. In total were 1934 of 1940 items recommended.



(b) BPR with MF on folds 1-5. In total were 1929 of 1940 items recommended.

Figure 5.6: Cumulative share of programs recommended with $N = 20$ for Weighted regularized matrix factorization and Bayesian personalized ranking with matrix factorization. A straight line indicates perfect diversity.

5.2 Looking for Context

Currently NRK TV only uses information of the triplets (*user*, *item*, *view*) from the dataset to give recommendations based on the WRMF model. However, NRK registers the timestamp of the event and which device the event occurred on, both of which could affect the users' taste in programs. Of the information variables registered by Google Analytics' cookies, device might be the most obvious contextual factor, on which the taste differs. It simply is more comfortable to watch long movies on TVs compared to the smaller screens of smart phones, which the user would have to hold by hand. However, the dataset does not have any way to track users across devices, so this section will focus on the time aspect.

Time has been one of the most researched contextual dimensions when it comes to RS. One famous instance of a temporal recommender system was the Netflix-prize winner known fittingly as *timeSVD++*, where time since the event was used to discount the importance of the events for users, and time was sorted into bins for items (Koren 2009). The name also references SVD, this is because the model minimizes the sum of squared errors, just like truncated SVD does. Shi et al. (2014) distinguishes between two types of temporal CF; *time-aware*, which are sensitive to a particular time line, and *time-dependent* CF, where there are cyclic phases. *Time-aware* RS will be able to discount the confidence in old events to allow for changing preferences, while *time-dependent* RS might recommend you an action movie on a Friday evening if that is what you would normally watch at that time. Only time-dependent RS use the entities $USER \times ITEM \times CONTEXT$ when it gives recommendations, what is called context aware and is the focus of this section. We will also look at a time-dependent model in the last section of the chapter.

We would like just enough levels of the context to make the best recommendations, too many will lead to more parameters to learn and fewer events for each context level. So even if in theory it would be best to divide context into many levels, we have to consider the data available to learn the parameters.

A good way of finding out which context categories to use, is to divide the time of events into bins in different ways, and look for bins with similar and different viewing patterns. Here we divide the events according to what weekday they occurred. More specifically we look at weekdays defined as the nychthemeron starting at 03:00 each morning. This is also how NRK defines the «TV days». How many events there has been on each weekday is shown in Figure 4.4. Figures 5.7, B.7 and B.8 show a visualization of how many events each genre/category has had in the dataset for each weekday. We note that Tuesday, Wednesday and Thursday behave very similarly for humor, news, documentary, entertainment and lifestyle. Monday and Friday are similar for humor, documentary and

lifestyle, while Saturday and Sunday look similar for humor and news. From this the context categories *MF* for Monday and Friday, *TWT* for Tuesday, Wednesday and Thursday, and *SS* for Saturday and Sunday, were chosen.

As mentioned above, by expanding the U-I matrix in another dimension (time), we end up with more data to feed to the model. For Section 5.3 only the 5,497 users with most events are considered. Even though working with tensors often put high demands on the hardware, this number could have been at least doubled without pushing the computers available, beyond their limits. The evaluation is done with holdout. The holdout consists of five folds, where in each fold a unique fifth of the users who have interacted with more than 20 unique programs/series have some randomly chosen events set to zero. More specifically each of the selected users first have five entries in the U-I-C tensor removed. Then, to be able to compare results across context categories and not just recommend items the user has interacted with in other contexts, the items were removed for all of the other context categories as well. So users can have 4 items removed if one item were removed for two different contexts in the first stage, and so on. Table 5.1 shows the relevant numbers. The bottom row shows the number of removed items in the preference matrix $\mathbf{Y}$, where $y_{ui} = 1$ if $y_{MF,ui} + y_{TWT,ui} + y_{SS,ui} > 0$ and 0 otherwise. So y_{ui} is missing if it was removed in all contexts it was nonzero in.

Table 5.1: Entries set to zero for context data.

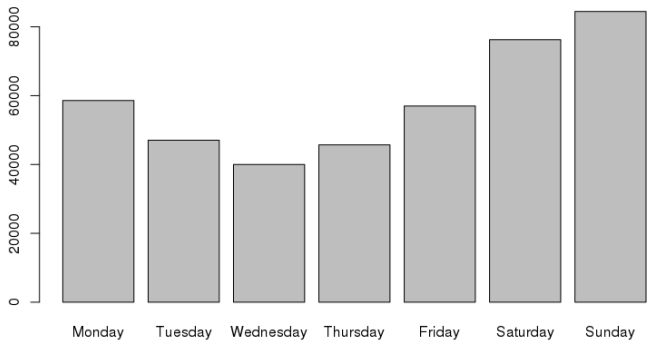
Context category	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Total
MF	2116	2106	2105	2135	2160	10622
TWT	2412	2399	2393	2396	2389	11989
SS	2176	2099	2095	2145	2141	10656
Across all	3848	3851	3829	3845	3847	19220

5.3 Context Aware Recommender Systems

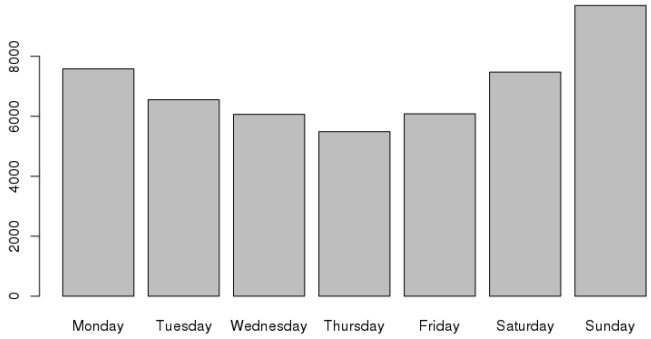
Now that we have found data with the context categories: MF, TWT and SS, it is time to compare the context aware RS described in Chapter 3. The first method calibrates the one model for each context. Later methods consists of one model for all contexts combined.

5.3.1 Weighted Regularized Matrix Factorization for Each Context

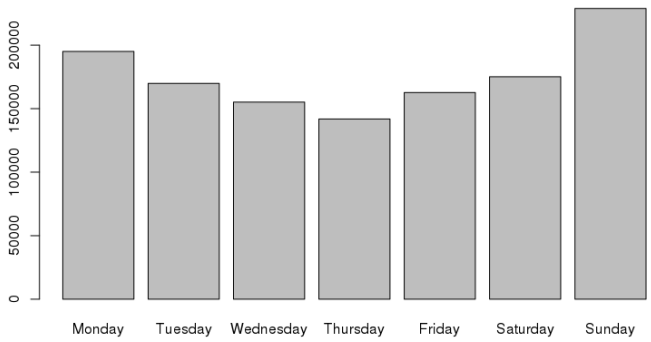
The simplest way of implementing context aware CF is to use the same model as before, but calibrate it for each context. For the WRMF model this requires us to calibrate the model for each of the three context levels. We have three



(a) Humor



(b) Movies



(c) Drama

Figure 5.7: Number of events for program categories across weekdays.

different user-item matrices, $\mathbf{R}_{MF}, \mathbf{R}_{TWT}$ and $\mathbf{R}_{SS}$, each with 5,497 users and 1,931 items. This way of doing context aware CF does require us to train the model three times and find the best combination of parameters for each. This is an example of pre-filtering, where the RS only uses information about the user's history with current context level to make recommendations (Campos et al. 2014). The «context aware» RS is then the combination of the three separate WRMF models.

The users in this smaller dataset are the ones with the most events. This should allow the model to learn the users' features well. The parameters are chosen as the combination that achieves the highest result for each context. For the level *MF* we get $\alpha_{MF} = 10, \lambda_{MF} = 0.015$ and $f_{MF} = 12$. Similarly we use $f_{TWT} = 15, f_{SS} = 12, \alpha_{TWT} = \alpha_{SS} = 7$ and $\lambda_{TWT} = \lambda_{SS} = 0.05$. These results can be found in Table A.2 in Appendix A.

5.3.2 Simple CANDECOMP/PARAFAC Decomposition

With the CP decomposition model of Secion 3.3.1 we can train the model directly on the data tensor $\underline{\mathbf{R}}$, without the need to split it into matrices for each context category. This should enable the model to *borrow strength* across contexts. This is not possible when we use one WRMF for each context category. Unfortunately, to our knowledge, there are no libraries available, for Python, Matlab or R, that does a regularized tensor decomposition, and therefore nor weighted regularized. After extensive research the author's conclusion is that there does not seem to be any freely available software that does weighted regularized tensor decomposition.

The results in this section are thus from simple CP decomposition, hence the title of this section, that solves the problem

$$\min_{\mathbf{X},\mathbf{Z},\mathbf{V}} \sum_{u,i,t} \left(y_{uit} - \sum_{r=1}^{f} x_{ur} z_{ir} v_{tr} \right)^2 .$$

This is comparable to a higher order SVD. One advantage of this is that we are only left with one parameter to choose, f the number of rank-1 tensors used in the approximation. After trying using both the preference tensor $\underline{\mathbf{Y}}$ and the user-item-context tensor $\underline{\mathbf{R}}$, using $\underline{\mathbf{Y}}$ gives far better results as can be seen by a quick glance at Figure 5.8. Therefore, the results presented in this section is with the binary user-item-context preference tensor. The alternating least squares algorithm for CP factorization from Matlab's Tensor toolbox by Bader et al. (2015) has been used for the computations. Because of the limitations the ALS algorithm used is not exactly the same as Algorithm 5, but the basic outline is similar to the factorization in Section 2.4.2. By unfolding the tensor $\underline{\mathbf{Y}}$ we can write

$$L(\mathbf{X},\mathbf{Z},\mathbf{V}) = \|\mathbf{Y}_{(1)} - \mathbf{X}(\mathbf{Z} \odot \mathbf{V})^T\|_F^2, \tag{5.3}$$

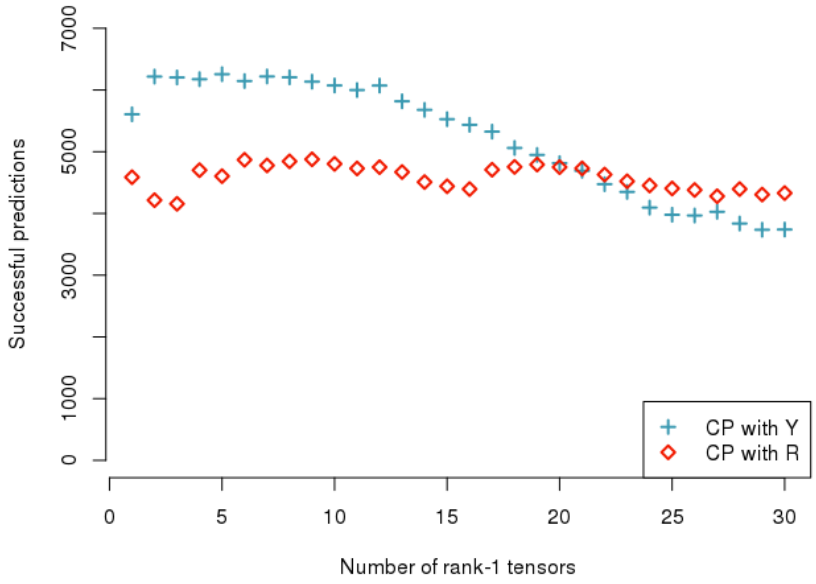


Figure 5.8: Results for CP using binary values **Y**, and implicit ratings **R**.

where $\mathbf{Y}_{(1)} \in \mathbb{R}^{U \times IT}$ is the mode-1 unfolded $\underline{\mathbf{Y}}$, and the latter term is the approximation. If we consider **Z** and **V** fixed and solve for **X** we get the closed form solution

$$\mathbf{X} = \mathbf{Y}_{(1)} \left((\mathbf{Z} \odot \mathbf{V})^{+} \right)^{T}, \tag{5.4}$$

where the superscript + indicates the pseudoinverse. In the same way, the closed form solutions for **Z** and **V** are

$$\mathbf{Z} = \mathbf{Y}_{(2)} \left((\mathbf{X} \odot \mathbf{V})^{+} \right)^{T} \tag{5.5}$$

$$\mathbf{V} = \mathbf{Y}_{(3)} \left((\mathbf{X} \odot \mathbf{Z})^{+} \right)^{T}. \tag{5.6}$$

So we get the solution for **X**, **Z** and **V** by repeating (5.4) - (5.6) until they convergence. More details can be found Tomasi & Bro (2006).

We can see from Figure 5.9 that the number of successes does not vary much between 2 and 12 rank-1 tensors. The highest number of successes is achieved with $f = 5$. With five rank-1 tensors the algorithm got 2,011 out of 10,622 for context *Monday-Friday*, 2,079 of 11,989 for *Tuesday-Wednesday-Thursday* and 2,166 of 10,656 for *Saturday-Sunday*, which gives a total of 6,256 out of 33,267 or 18.8 %. As noted above there are several values of f that are almost as good and $f = 7, 2, 8, 3$, reported in decreasing order of success, all result in more than 6,200 successes.

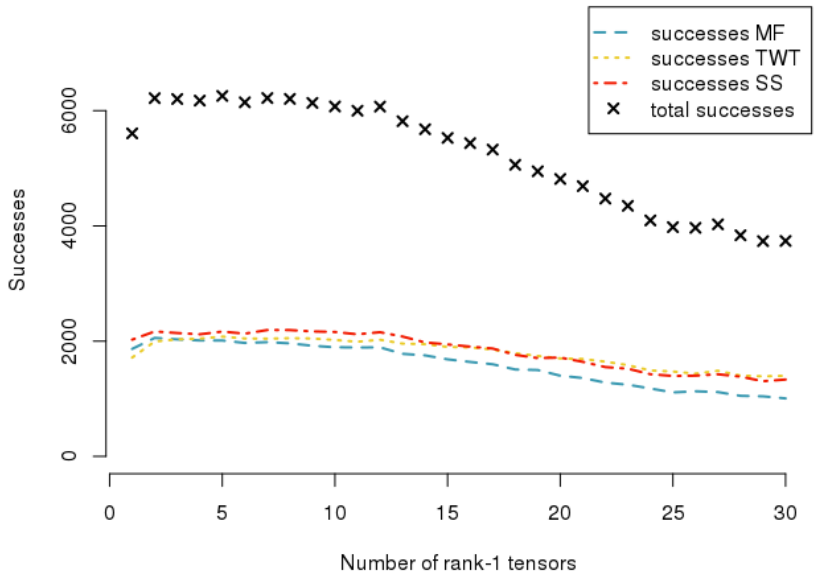


Figure 5.9: Results from holdout for CP decomposition

5.3.3 Simple Tucker Decomposition

The next context aware model uses the Tucker decomposition. We recall that Tucker decomposition factorizes the tensor into three matrices: $\mathbf{X}, \mathbf{Z}, \mathbf{V}$ and one core tensor $\underline{\mathbf{S}}$. As with the CP decomposition in the previous section, we know of no implementation of the Weighted regularized Tucker decomposition model. This is less surprising as there is no literature, known to the author, about it. The problem Tensor toolbox' alternating least squares algorithm solves is

$$\min_{\mathbf{X},\mathbf{Z},\mathbf{V},\underline{\mathbf{S}}} \sum_{u,i,t} \left(y_{uit} - \sum_{a=1}^{d_X} \sum_{b=1}^{d_Z} \sum_{c=1}^{d_V} s_{abc} x_{ua} z_{ib} v_{tc} \right)^2.$$

Even though we do not have the regularization terms we still have to find the size of the core matrix, namely find d_X, d_Z and d_V. As the CP decomposition is a restricted version of the Tucker decomposition, we would assume that Tucker performs better. It has the core tensor $\underline{\mathbf{S}}$ that allows for more intricate interplay between user, item and context category. In the same way as SVD and matrix factorization are not unique, we can show that the Tucker decomposition has the same property. Let $\mathbf{G}$ be an orthogonal $d_X \times d_X$ matrix. We can write

$$\underline{\mathbf{A}} = (\underline{\mathbf{S}} \times_1 \mathbf{G}^T) \times_1 (\mathbf{XG}) \times_2 \mathbf{Z} \times_3 \mathbf{V} = \underline{\mathbf{S}} \times_1 (\mathbf{XGG}^T) \times_2 \mathbf{Z} \times_3 \mathbf{V} = \underline{\mathbf{S}} \times_1 \mathbf{X} \times_2 \mathbf{Z} \times_3 \mathbf{V},$$

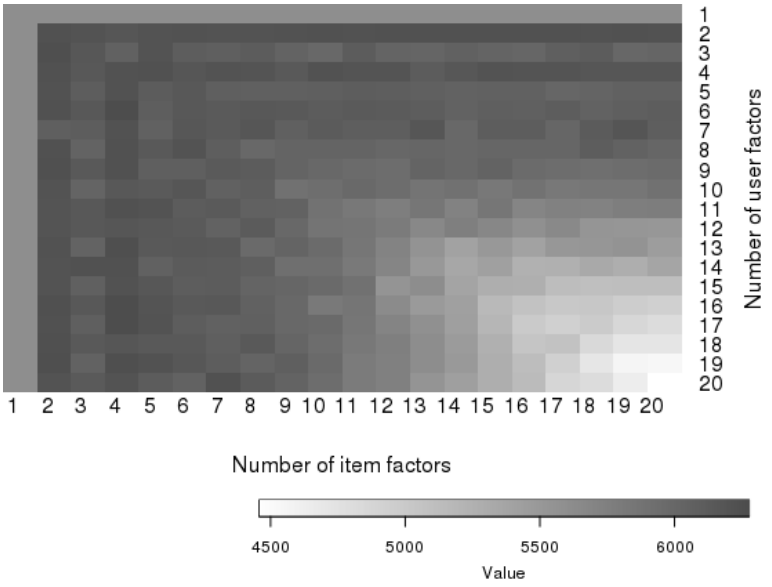


Figure 5.10: Heatmap of total number of successful predictions for simple Tucker decomposition for different values of item factors d_Z and user factors d_X, while $d_V = 1$.

where we have used that for mode-n multiplication we have $\underline{\mathbf{A}} \times_n \mathbf{B} \times_n \mathbf{C} = \underline{\mathbf{A}} \times_n (\mathbf{CB})$. This rotation and scaling can of course also be done to the other matrices as well. So without regularization we can expect the model to have issues with the scaling in some feature entries.

The heatmap in Figure 5.10 shows the total number of successes, aggregated over the three contexts, achieved in the holdout procedure for $d_V = 1$. The upper limit of d_V is 3 since optimal V is the d_V left leading singular values of the matrix $\mathbf{Y}_{(3)}(\mathbf{X} \otimes \mathbf{Z})$ with three rows and $d_X d_Z$ columns (Kolda & Bader 2009). The two heatmaps for $d_V = 2$ and $d_V = 3$, Figures B.9 and B.10 can be found in the Appendix B. The heatmaps show that the best results are achieved with a higher number of item factors compared to user factors, this most evident for $d_V = 2$ and $d_V = 3$. The top 10 combinations of number of factors and successes are presented in Table 5.2. We see that two combinations of parameters score equally well and neither of them seems to be an outlier when we look at the rest of the table. The one in the first row has a lower number of parameters on users, items and contexts and thus we have $2U + 3I + T + 239$, here 17,029, less factors to compute and store in memory. In total we go from a tensor with 31,844,121 entries to an approximation made up by 58,547 entries, which is close to just 0.2 % of the original number of entries.[2]

[2]The first number is $5,497*1,931*3$. The second number is $5*5,497+16*1,931+2*3+5*16*2$.

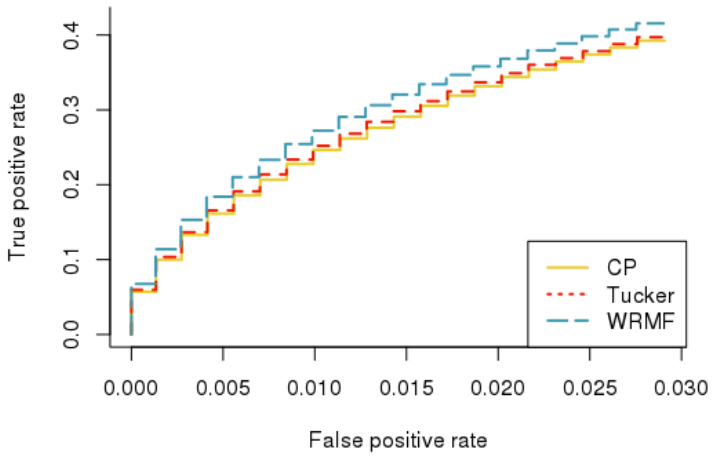


Figure 5.11: Partial ROC curves for «context aware» WRMF, CP and Tucker decomposition with $N = 0, 1, ..., 20$

Table 5.2: Results for simple Tucker decomposition with 5-fold holdout.

User factors	Item factors	Context factors	MF	TWT	SS	Total successes
5	16	2	2038	2118	2252	6408
7	19	3	2009	2114	2285	6408
5	12	3	2042	2105	2252	6399
4	18	2	2038	2086	2273	6397
5	17	3	2080	2078	2238	6396
4	11	2	2047	2088	2256	6391
4	15	3	2050	2081	2258	6389
4	9	2	2050	2075	2261	6386
4	20	2	2046	2080	2260	6386
4	17	3	2053	2082	2249	6384

In Figure 5.11 we can see the partial ROC curves for the context aware WRMF, the simple CP decomposition and the simple Tucker decomposition. The figure shows that WRMF has a higher true positive rate for all values of false positive rate. The two tensor models have very similar ROC curves, but the Tucker model does a little better. However, we can not just give up on the tensor models. As explained above, they lack regularization and the use of weights. The users in the dataset are also the ones with most events so the advantage of borrowing strength across context categories might be small. We will look at how much adding a regularization parameter and weights to the sum of squared error improves recommendations.

5.3.4 Singular Value Decomposition for Contexts

As explained above, the baseline model, WRMF, has the advantage of regularization to prevent overfitting, and weights to fit the features better to event we are more certain of. WRMF performed better for all N tested. There could be several explanations to this. The model could fit the data better, the levels of context could be irrelevant *or* the lack of confidence weights and regularization is hurting the tensor models. The last explanation will be tested in this section by removing confidence weights and regularization from the WRMF model. We are then left with finding the truncated singular value decomposition. Singular value decomposition is closely related to both CP and the Tucker decomposition as discussed in chapter 2. Because of this, comparing truncated SVD to the weighted and regularized WRMF model, should give an indication of the performance boost one would expect from adding regularization and weights to the model.

All popular statistical software have a way of doing singular value decomposition. Because of the rather small dataset considered here, the standard *svd*-function in R was used for the computations in this section. The highest number of successful recommendations was achieved using just 2 singular values for context *MF* and *SS*, while using 5 for *TWT*. The results did not vary much for singular values from 2 to 10. Using the best combination gives us the partial ROC curve in Figure 5.12. The curves for WRMF and CP decomposition are also drawn in the figure. We see that the lines for SVD and CP lie close to each other. Truncated SVD does better for few recommendations, then they cross at $N = 12$ and after this CP does better. If we had a similar boost in performance for CP as WRMF has from truncated SVD, we would expect the tensor models to have similar results as WRMF.

The results presented in this section were not what the author had wanted to present. There could be several causes for the poor performance of the context aware tensor models. One of the more obvious being the context categories. Ideally, the users preferences changes from one category to the next, but here

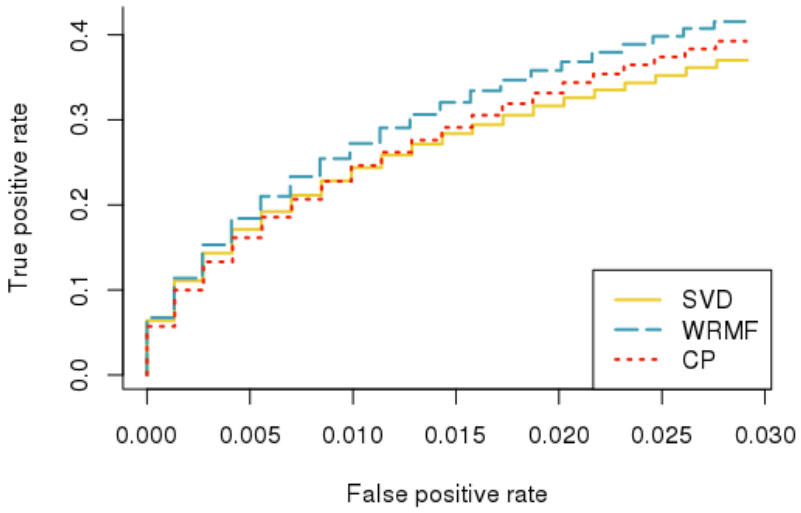


Figure 5.12: Partial ROC curves for truncated SVD, WRMF and CP decomposition

it might be that they vary more inside the category than across categories. It would be exciting to test if splitting for devices could yield better results, or go even simpler with day-time vs night-time. This is not pursued here, but the same framework could be applied.

5.4 Temporally Weighted Regularized Matrix Factorization

While the time might not yet be ripe for weighted regularized tensor decomposition, there are other ways to utilize the timestamps of the events. We can allow for preferences to change over time with a time-aware RS. The dataset only consists of events from a time span of less than a year, 308 days, but this is a stone that should be turned nonetheless. When we allow for users' taste to change we believe that a user could be different at the start compared to the end of the time interval. Because we want to give good recommendations now and in the future, we are more interested in the users preferences now. One easy way of accomplishing this is by giving higher weights to newer observations in the loss function. We can implement this by adjusting the confidence function from (3.5) to allow for time since last event with the item. The new confidence function is

$$c' = c(r,t) = 1 + \alpha r + \beta \Phi\left(\frac{t-\bar{t}}{\sigma}\right), \tag{5.7}$$

where β is a parameter that determines the importance of time deviance in confidence and $\Phi(\cdot)$ denotes the standard normal cumulative function. The term t is the date of the event represented as the number of days since the first event in the dataset. $\bar{t}$ is the mean of all the user's ts, i.e. the user's mean date of events. Lastly, σ is the user's empirical standard deviation of his ts. The new loss function becomes

$$\min_{\mathbf{X}\in\mathbb{R}^{U\times f},\mathbf{Z}\in\mathbb{R}^{I\times f}} \sum_{u,i} c'_{ui,t}(y_{ui}-\mathbf{x}_u^T\mathbf{z}_i)^2+\lambda\left(\sum_u\|\mathbf{x}_u\|^2+\sum_i\|\mathbf{z}_i\|^2\right).$$

It is very similar to (3.6), the only difference is the term $c'_{ui,t}$ which highlight the time dependency of the weight.

The confidence function (5.7) will allow for user preferences to drift. So if a user has had a hangup on war dramas two years ago, but got enough of it and has moved on to lighter content he would not want the RS to keep recommending him more war dramas. This quality could also be a big plus for younger users who mature and therefore change their taste in television. Another simple solution to this is to only use data from the last year or so. We could also modify the confidence function so it gives higher weight to other periods. One such period where the taste might change is Christmas. If we have observations for last Christmas we can give these higher weights to get the users in the right holiday spirit. We can also use this to «remind» users of what they used to be into. Here we will continue to use the confidence function that gives newer events higher weights.

Temporally WRMF will, however, also have the tendency to recommend items that other users have watched more recently. This could be beneficial, but online streaming services like NRK TV often have a separate row with items that are popular at the time. For NRK this is especially important. NRK has current programs like *Nytt på nytt* and *Dagsrevyen* that are most relevant directly after they are produced, but they also have series and movies that only are available for some weeks. Both *Nytt på nytt* and *Dagsrevyen* are released regularly and because the U-I matrix $\mathbf{R}$ is created on a series level and does not include an entry for each episode, as explained earlier, the Temporally WRMF model should not have a significant advantage in predicting these kinds of programs. In the dataset described in Chapter 4, the entries that are set to zero for testing are the the user's last events. We would expect the Temporally WRMF to recommend more of NRK's content that are only available in the later part of the time span while WRMF would recommend more of content that had already expired. By introducing a temporal effect on items as well, we could remove the models property of recommending new items by adding a parameter to (5.7) that controls for days since release or time since mean date.

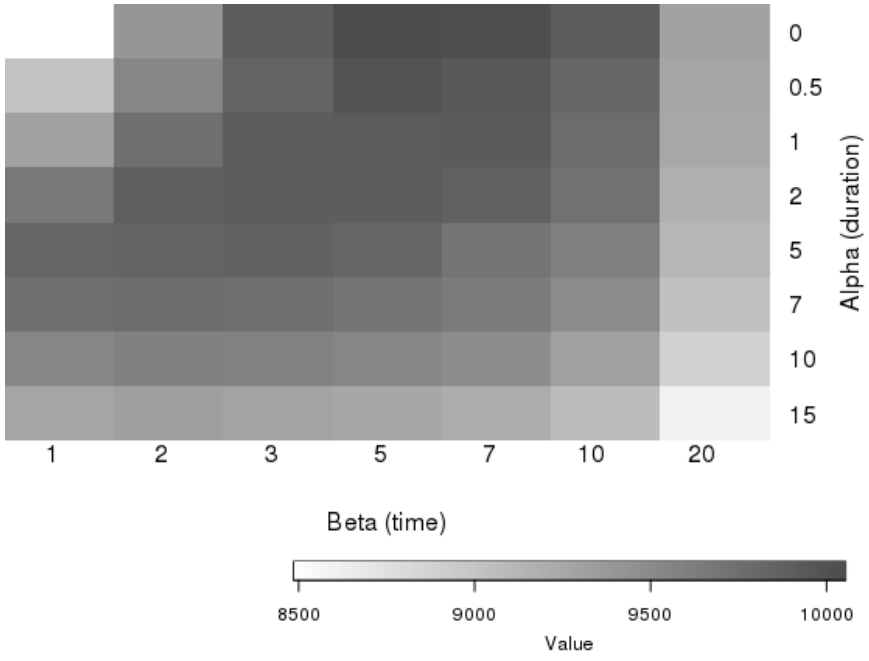


Figure 5.13: Results from holdout for Temporally WRMF with $f = 10$

The model has been applied on the bigger dataset from Section 5.1 with 35,276 users and 1,940 programs. We have the same five folds, where in each fold a unique set of users have had five items they have interacted with removed. For each user we recommend the 12 items which vectors have the highest dot product with the user's feature vector, that he has not interacted with in the training data. Successful predictions are then that the removed items are in the recommended list. The results presented are then summed over the folds. Because of the way the test data is created the model should perform better than the baseline WRMF, which is a special case with $\beta = 0$, and the first indication is that it does. Figure 5.13 shows a heatmap of successful recommendations with $f = 10$ and best regularization parameter λ chosen for each tile. We remember that the best result from Section 5.1 was 9701. The Temporally WRMF has a significant increase in performance compared to WRMF. The highest score is 10,054. While this is seems very good, a quick look at Figure 5.13 shows that the best results are achieved with very low value of α. This could be because user's taste evolves quite a lot. The consequence of low α and a higher β is that programs that have been watched recently are fitted better and that the implicit rating, how long the user has interacted with the item, is less important. Programs that are only watched in the beginning of the period will thus have feature vectors with entries close to zero because of the regularization parameter. As we can easily visualize with the help of Figure 2.1, programs with more

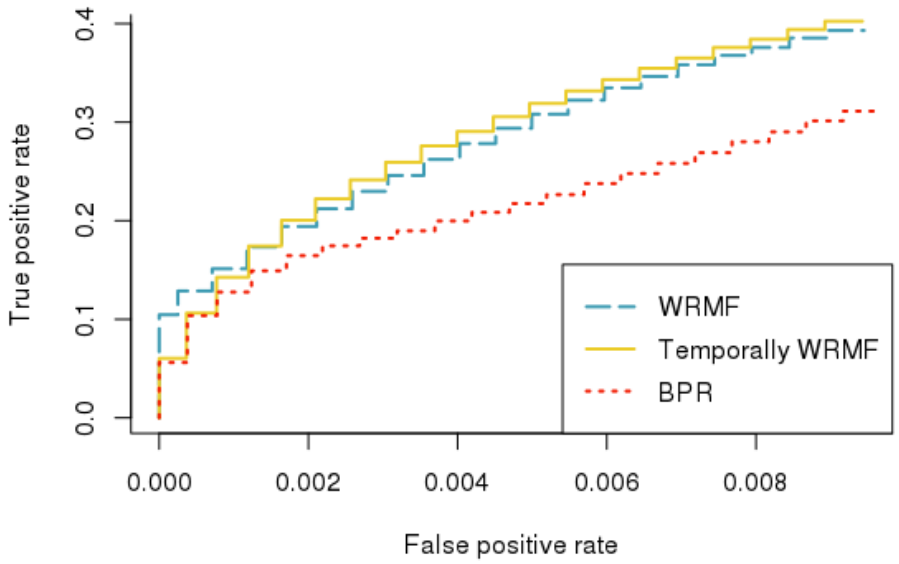


Figure 5.14: Partial ROC curves for WRMF, BPR with MF and Temporally WRMF with $N = 0, 1, ..., 20$.

extreme feature values will be more likely to be recommended because user u's predicted preference for item i is the inner product of the feature vectors.

Users will now be recommended the items that fit their feature vector *and* are popular at the time. This result is not very surprising given that the start page of NRK TV has shown a list of popular and new content in the time period of which the data is taken from. The figure only shows results for $f = 10$, but the 26 best combinations tried are all with $f = 10$ and anyway it gives a good picture of the results. The hypothesis that how long you interact with the item is not positively correlated to how much you like it does not sound plausible. The idea of preferences being so time dependent does not sound true either. What seems like the most likely explanation is exactly that many users watch the same new, popular programs. Because the idea that α should be zero seems unreasonable, the parameters chosen to construct the ROC curve were the combination $\alpha = 2$ and $\beta = 3$ for the confidence function, $f = 10$ and $\lambda = 0.005$. This should allow for drifting preferences, while still giving more weight to user-item combinations with high implicit ratings. The parameter combination chosen still did better than WRMF with 9,913 successes for $N = 12$.

In Figure 5.14 we see the partial ROC curves for Temporally WRMF, WRMF and

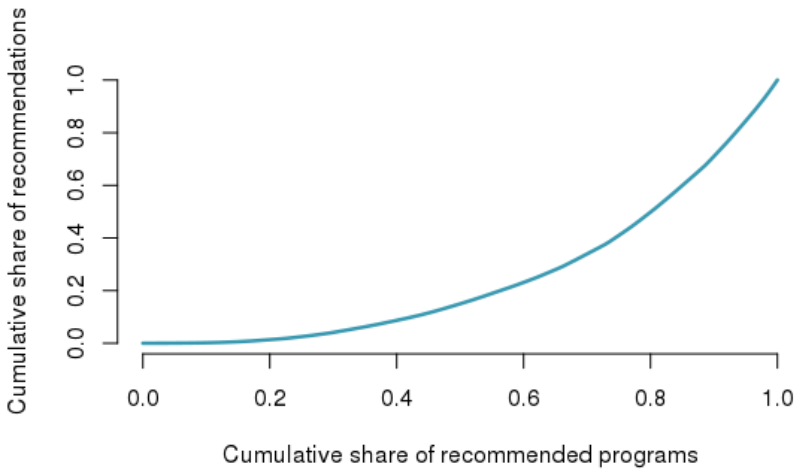


Figure 5.15: Cumulative share of programs recommended with $N = 20$ for Temporally weighted regularized matrix factorization on folds 1-5. A straight line indicates perfect diversity for recommended items. Only 289 items were recommended in total.

BPR with MF. While traditional WRMF does better for the lowest values of false positive rate, Temporally WRMF surpasses it and does better for the remaining interval. NRK TV are also interested in how diverse the recommendations are. How well Temporally WRMF does in this regards is shown in Figure 5.15. We note that only 289 out of 1,940 items in the dataset are recommended when $N = 20$, so even though the graph makes it look like it does better than WRMF at first glance, it only recommends 15 % of the programs WRMF does. Some of the programs that does not get recommended have likely expired, but the number is still low.

While Temporally WRMF arguably gives better recommendations than WRMF based on the ROC curves, its recommendations are less diverse. It has a tendency to recommend what is popular at the time. The «problem» is, as previously mentioned, that most services already provide a list of popular items and that the RS is supposed to supplement this. If the service did not have such a list of the popular items the Temporally WRMF could offer a significant improvement to regular WRMF.

Chapter 6

Discussion

In this thesis I have reviewed both traditional collaborative filtering methods using matrix factorization, and state of the art methods using tensor decomposition. The methods have been tested on a dataset from NRK TV. The goal has been to find or develop a recommender system that gives better recommendations than the weighted regularized matrix factorization (WRMF) model currently used by NRK TV. This has been done by trying to utilize more of the information available. This goal was somewhat achieved with the temporal WRMF of Section 5.4, where we allow user's taste to change with time by giving higher weight to newer events. This model performed better with the objective to predict the newest, removed events. Bringing time into the confidence function can open up for new possibilities. If we put more weight on older events we could get recommendations of items similar to what the user used to interact with. Over time this could be used on holidays such as Christmas, where one could think that taste differs.

The context aware tensor models were only tested without using weights and regularization. This is a severe handicap and the relatively poor performance is believed caused by this. The context categories in this thesis were most likely not optimally chosen. It might have been that the results would be more in the context aware models favor if they were chosen in a better way. NRK TV has recently, after the time period of the dataset used here, started allowing users to log into the service with a user ID. This will allow for tracking users across devices. It is the belief of the author that this might be a more reasonable variable to divide context by. Some of the technical aspects of creating a context aware recommender system have been introduced in Chapter 3 and could be used for later work to build upon, even though the implementation is missing.

One very interesting property of the NRK TV dataset is the extreme outlier Skam. It is by far the most popular item in the dataset and was a national phenomenon in the time period considered. Future work could be to investigate how performance is dependent on Skam. Perhaps it should be weighted differently

because everyone would have heard of it or because everyone watches it to be able to join discussions in social events. By being such an extreme outlier it amounts to a significant part of the total weights in the WRMF model, and it would be interesting to see what would happen if we removed events for Skam.

If possible, it could be beneficial for NRK TV to register which programs the user have scrolled through while on their «home screen». Giving different weights to these negative events, that the user at least knows something about but chooses not to click on, could result in better recommendations. Luckily for the users of NRK TV, the conclusion of this thesis is that WRMF does a good job.

Bibliography

Adomavicius, G. & Tuzhilin, A. (2015), Context-aware recommender systems, *in* 'Recommender systems handbook', Springer, pp. 191–226.

Ahn, H. J. (2008), 'A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem', *Information Sciences* **178**(1), 37–51.

Bader, B. W., Kolda, T. G. et al. (2015), 'Matlab tensor toolbox version 2.6', Available online.
URL: *http://www.sandia.gov/ tgkolda/TensorToolbox/*

Buchanan, A. M. & Fitzgibbon, A. W. (2005), Damped newton algorithms for matrix factorization with missing data, *in* 'Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on', Vol. 2, IEEE, pp. 316–322.

Campos, P. G., Díez, F. & Cantador, I. (2014), 'Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols', *User Modeling and User-Adapted Interaction* **24**(1-2), 67–119.

Carroll, J. D. & Chang, J.-J. (1970), 'Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition', *Psychometrika* **35**(3), 283–319.

Cremonesi, P., Turrin, R., Lentini, E. & Matteucci, M. (2008), An evaluation methodology for collaborative recommender systems, *in* 'Automated solutions for Cross Media Content and Multi-channel Distribution, 2008. AXMEDIS'08. International Conference on', IEEE, pp. 224–231.

Davis, J. & Goadrich, M. (2006), The relationship between precision-recall and roc curves, *in* 'Proceedings of the 23rd international conference on Machine learning', ACM, pp. 233–240.

Eckart, C. & Young, G. (1936), 'The approximation of one matrix by another of lower rank', *Psychometrika* **1**(3), 211–218.

Faber, N. K. M., Bro, R. & Hopke, P. K. (2003), 'Recent developments in candecomp/parafac algorithms: a critical review', *Chemometrics and Intelligent Laboratory Systems* **65**(1), 119–137.

Frederickson, B. (2017), *implicit: Collaborative Filtering for Implicit Datasets*. Python module version 0.1.5.
URL: *http://github.com/benfred/implicit/*

Funk, S. (2006), 'Netflix update: Try this at home', Available online.
URL: *http://sifter.org/ simon/journal/20061211.html*

Harshman, R. A. (1970), 'Foundations of the parafac procedure: Models and conditions for an" explanatory" multi-modal factor analysis'.

Hidasi, B. & Tikk, D. (2012), Fast als-based tensor factorization for context-aware recommendation from implicit feedback, *in* 'Joint European Conference on Machine Learning and Knowledge Discovery in Databases', Springer, pp. 67–82.

Hu, Y., Koren, Y. & Volinsky, C. (2008), Collaborative filtering for implicit feedback datasets, *in* '2008 Eighth IEEE International Conference on Data Mining', Ieee, pp. 263–272.

Johnson, C. C. (2014), Logistic matrix factorization for implicit feedback data, *in* 'NIPS 2014 Workshop on Distributed Machine Learning and Matrix Computations'.

Karatzoglou, A., Amatriain, X., Baltrunas, L. & Oliver, N. (2010), Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering, *in* 'Proceedings of the fourth ACM conference on Recommender systems', ACM, pp. 79–86.

Kiers, H. A. & Mechelen, I. V. (2001), 'Three-way component analysis: Principles and illustrative application.', *Psychological methods* **6**(1), 84.

Kolda, T. G. (2001), 'Orthogonal tensor decompositions', *SIAM Journal on Matrix Analysis and Applications* **23**(1), 243–255.

Kolda, T. G. (2006), *Multilinear operators for higher-order decompositions*, United States. Department of Energy.

Kolda, T. G. & Bader, B. W. (2009), 'Tensor decompositions and applications', *SIAM review* **51**(3), 455–500.

Koren, Y. (2009), 'The bellkor solution to the netflix grand prize', *Netflix prize documentation* **81**, 1–10.

Kruskal, J. (1989), Rank, decomposition, and uniqueness for 3-way and n-way arrays, *in* 'Multiway data analysis', North-Holland Publishing Co., pp. 7–18.

Lam, X. N., Vu, T., Le, T. D. & Duong, A. D. (2008), Addressing cold-start problem in recommendation systems, *in* 'Proceedings of the 2nd international conference on Ubiquitous information management and communication', ACM, pp. 208–211.

Lay, D. C. (2002), 'Linear algebra and its applications'.

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

Rendle, S., Freudenthaler, C., Gantner, Z. & Schmidt-Thieme, L. (2009), Bpr: Bayesian personalized ranking from implicit feedback, *in* 'Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence', AUAI Press, pp. 452–461.

Rendle, S. & Schmidt-Thieme, L. (2010), Pairwise interaction tensor factorization for personalized tag recommendation, *in* 'Proceedings of the third ACM international conference on Web search and data mining', ACM, pp. 81–90.

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P. & Riedl, J. (1994), Grouplens: an open architecture for collaborative filtering of netnews, *in* 'Proceedings of the 1994 ACM conference on Computer supported cooperative work', ACM, pp. 175–186.

Salakhutdinov, R. & Mnih, A. (2007), Probabilistic matrix factorization., *in* 'Nips', Vol. 1, pp. 2–1.

Sarwar, B., Karypis, G., Konstan, J. & Riedl, J. (2001), Item-based collaborative filtering recommendation algorithms, *in* 'Proceedings of the 10th international conference on World Wide Web', ACM, pp. 285–295.

Schwartz, B. (2004), 'The paradox of choice: Why less is more', *New York: Ecco* .

Shani, G. & Gunawardana, A. (2011), Evaluating recommendation systems, *in* 'Recommender systems handbook', Springer, pp. 257–297.

Shi, Y., Larson, M. & Hanjalic, A. (2014), 'Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges', *ACM Computing Surveys (CSUR)* **47**(1), 3.

Tomasi, G. & Bro, R. (2006), 'A comparison of algorithms for fitting the parafac model', *Computational Statistics & Data Analysis* **50**(7), 1700–1734.

Tucker, L. R. (1966), 'Some mathematical notes on three-mode factor analysis', *Psychometrika* **31**(3), 279–311.

Wang, X. & Navasca, C. (2015), 'Convergence and acceleration of the regularized alternating least square algorithm for tensor approximation', *arXiv preprint arXiv:1507.04721* .

Wright, S. & Nocedal, J. (1999), 'Numerical optimization', *Springer Science* **35**, 67–68.

Yates, F. (1933), 'The analysis of replicated experiments when the field results are incomplete', *Empire Journal of Experimental Agriculture* **1**(2), 129–142.

Çoba, L. (2016), *rrecsys: Environment for Assessing Recommender Systems*. R package version 0.9.5.4.
URL: *https://CRAN.R-project.org/package=rrecsys*

Appendix A

Tables

Table A.1: Top 10 results of BPR-MF 5-fold holdout validation with learning rate, $\alpha = 0.025$.

Factors (f)	Regularization (λ)	Successes
50	0.025	7335
5	0.100	7334
85	0.015	7321
85	0.005	7298
30	0.015	7280
5	0.010	7277
40	0.050	7277
150	0.100	7273
150	0.025	7262
30	0.005	7236

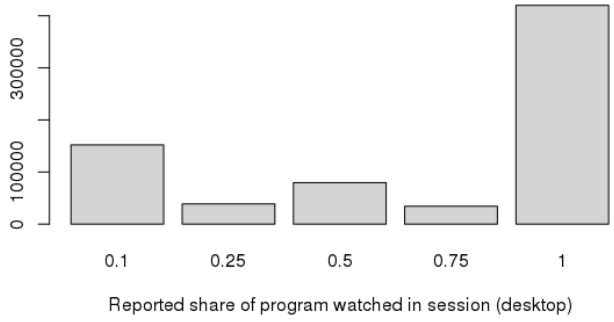
Table A.2: Top 5 results of WRMF for context with 5-fold holdout validation.

Context	Factors (f)	Regularization (λ)	Confidence (α)	Successes
MF	12	0.015	10	2239
	10	0.025	10	2238
	10	0.010	10	2234
	10	0.100	10	2231
	10	0.015	10	2210
TWT	15	0.050	7	2435
	12	0.005	7	2428
	15	0.005	5	2422
	12	0.015	5	2421
	15	0.010	10	2420
SS	12	0.050	7	2444
	12	0.100	10	2433
	12	0.025	7	2431
	12	0.050	10	2431
	10	0.005	10	2430

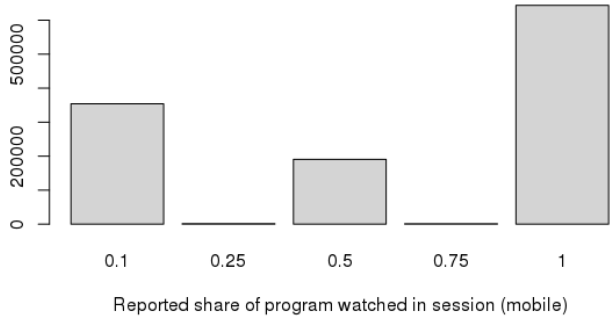
Appendix B

Figures

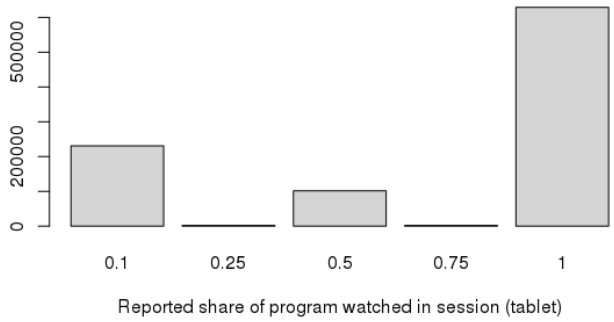
B.1 Figures for Chapter 4



(a) Share from desktop

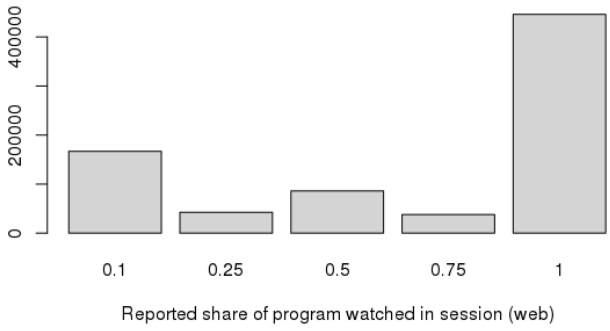


(b) Share mobile

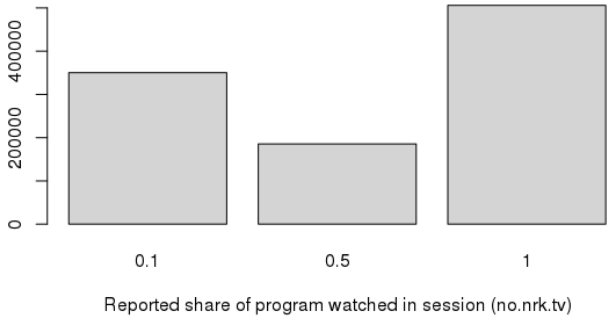


(c) Share tablet

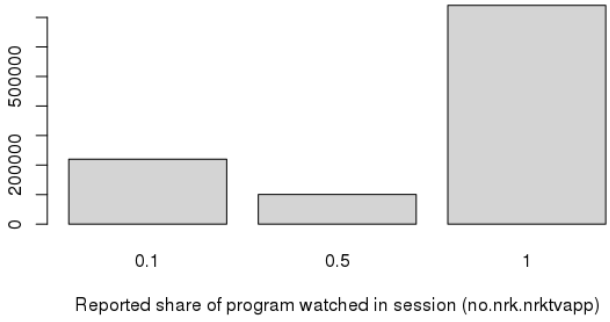
Figure B.1: Shares of events device



(a) Share web

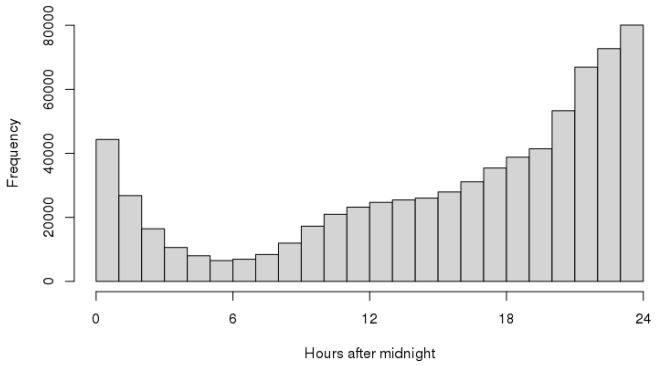


(b) Share no.nrk.tv

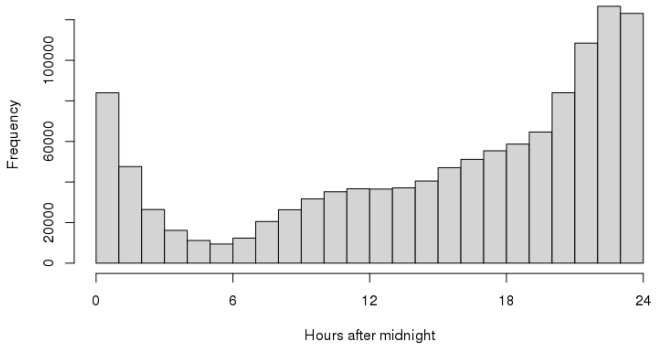


(c) Share no.nrk.nrktvapp

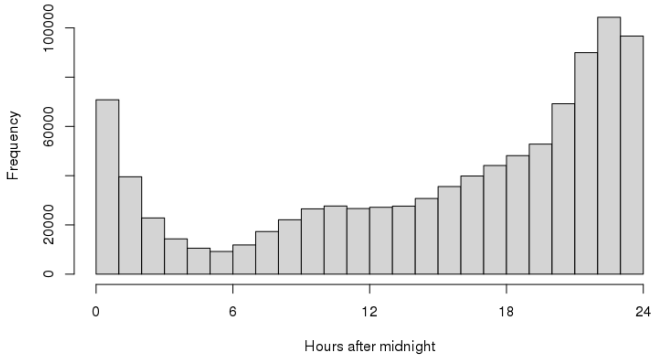
Figure B.2: Shares of events app



(a) Hours desktop

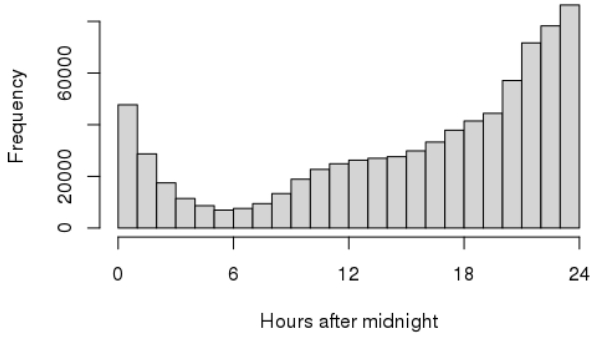


(b) Hours mobile

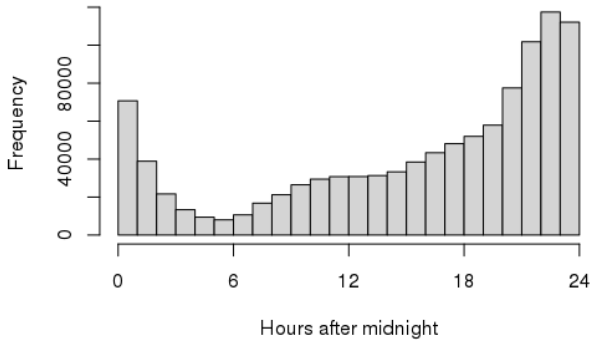


(c) Hours tablet

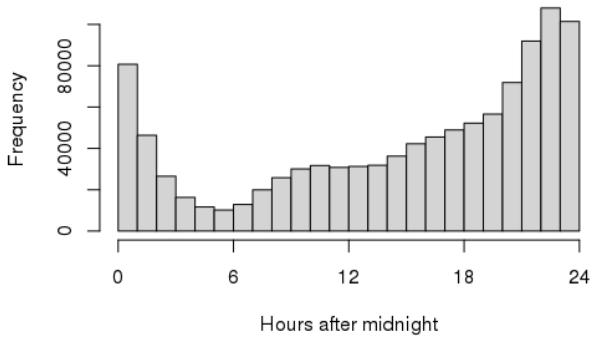
Figure B.3: Hours of events for devices



(a) Hours web

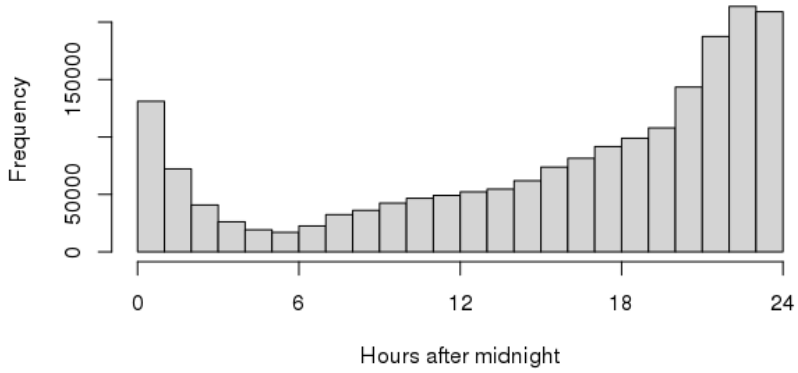


(b) Hours no.nrk.tv

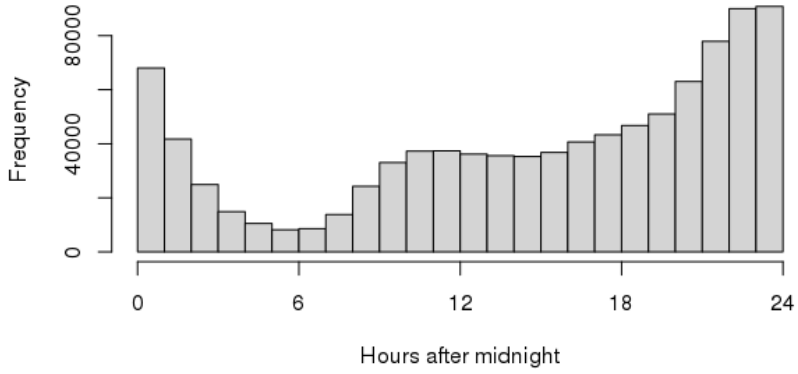


(c) Hours nonrknrktvapp

Figure B.4: Hours of events for app IDs



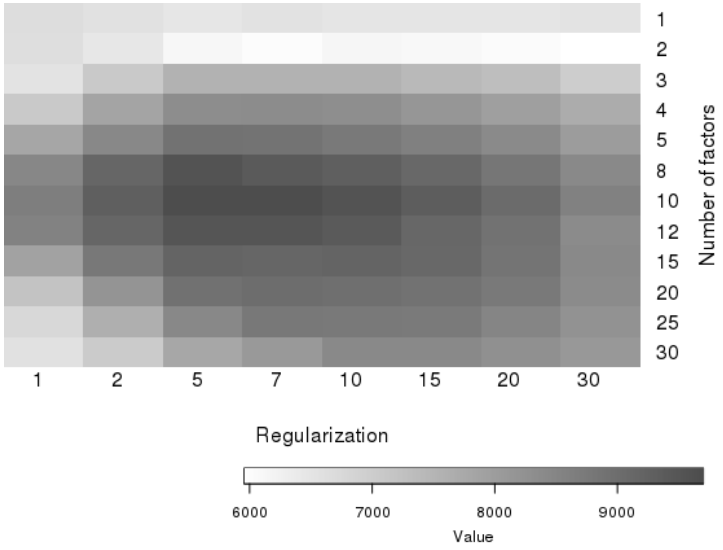
(a) Hours of events weekdays (Monday - Friday).



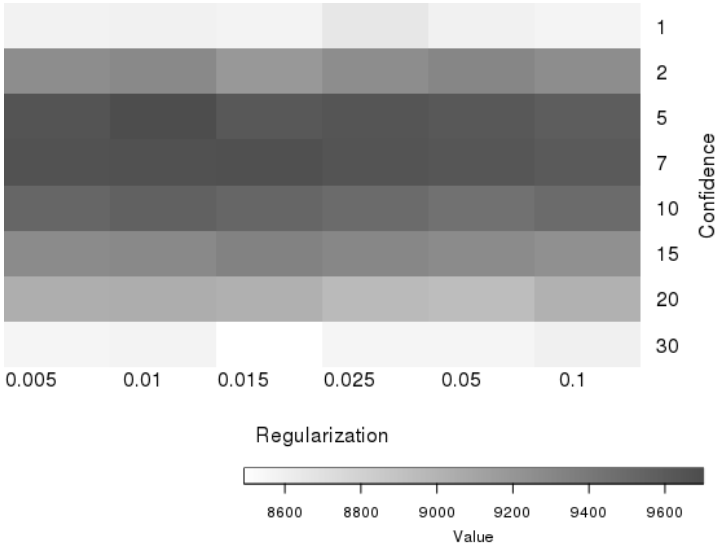
(b) Hours of events weekend (Saturday - Sunday).

Figure B.5: Hours of events 2

B.2 Figures for Chapter 5

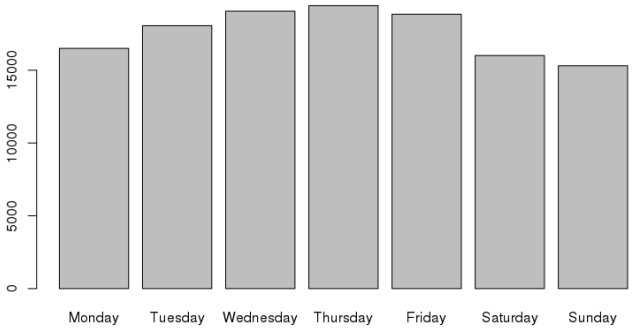


(a) WRMF results for combinations of number of factors and confidence parameter

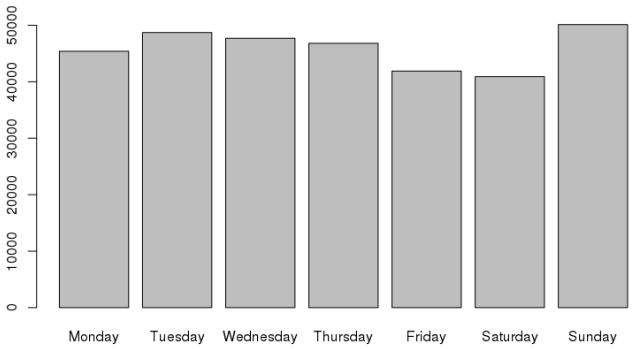


(b) WRMF results for combinations of regularization and confidence parameter

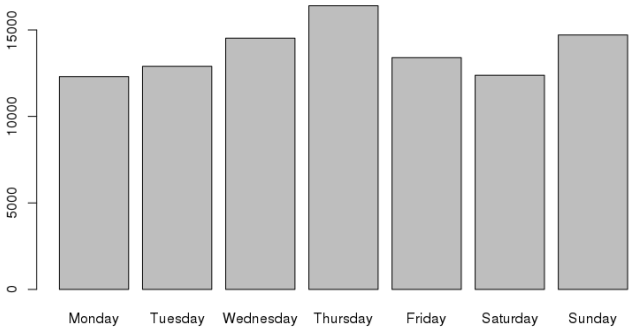
Figure B.6: Heatmaps for WRMF. Each tile is chosen as the maximum



(a) News

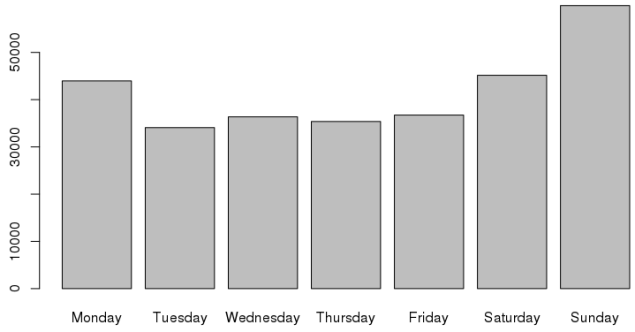


(b) Documentary

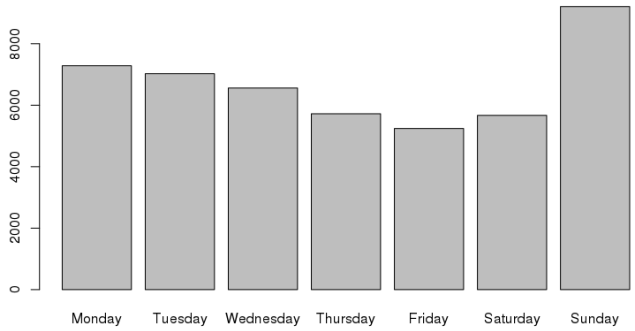


(c) Science

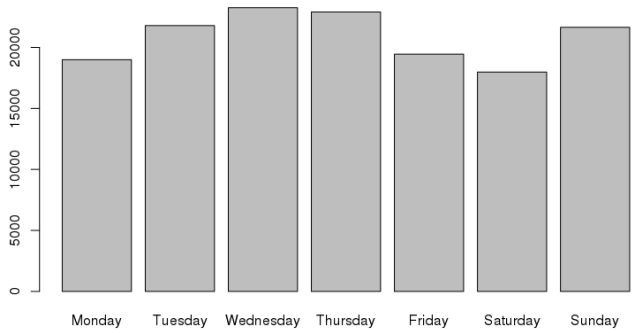
Figure B.7: Number of events for program categories across weekdays.



(a) Entertainment



(b) Nature



(c) Lifestyle

Figure B.8: Number of events for program categories across weekdays.

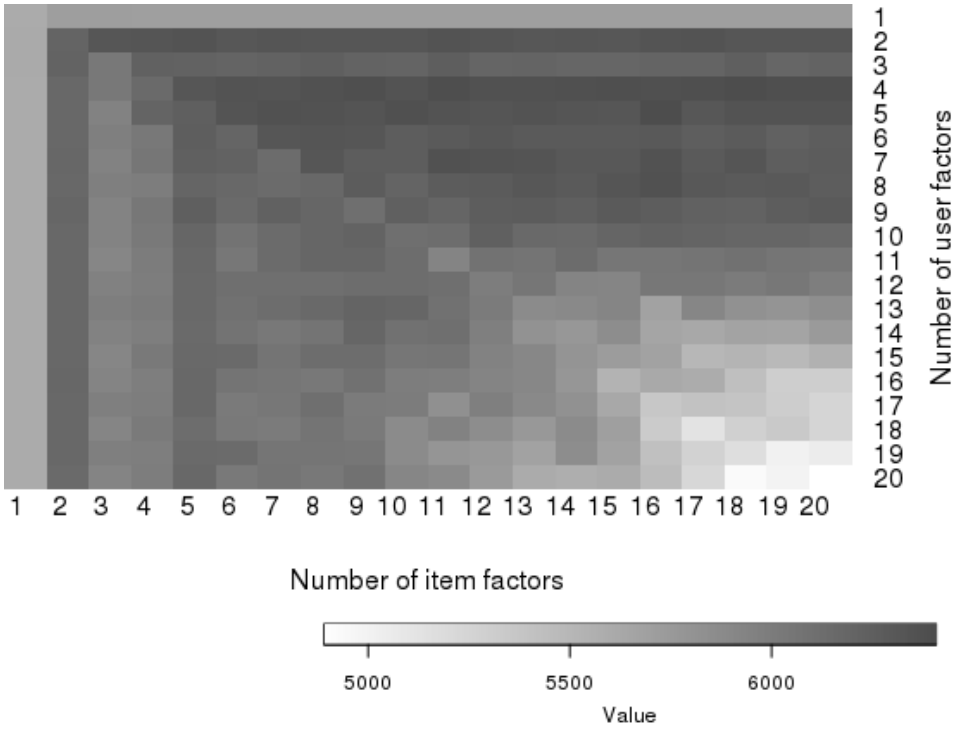


Figure B.9: Heatmap of total number of successes across all three contexts for Tucker3 decomposition with 2 item factors

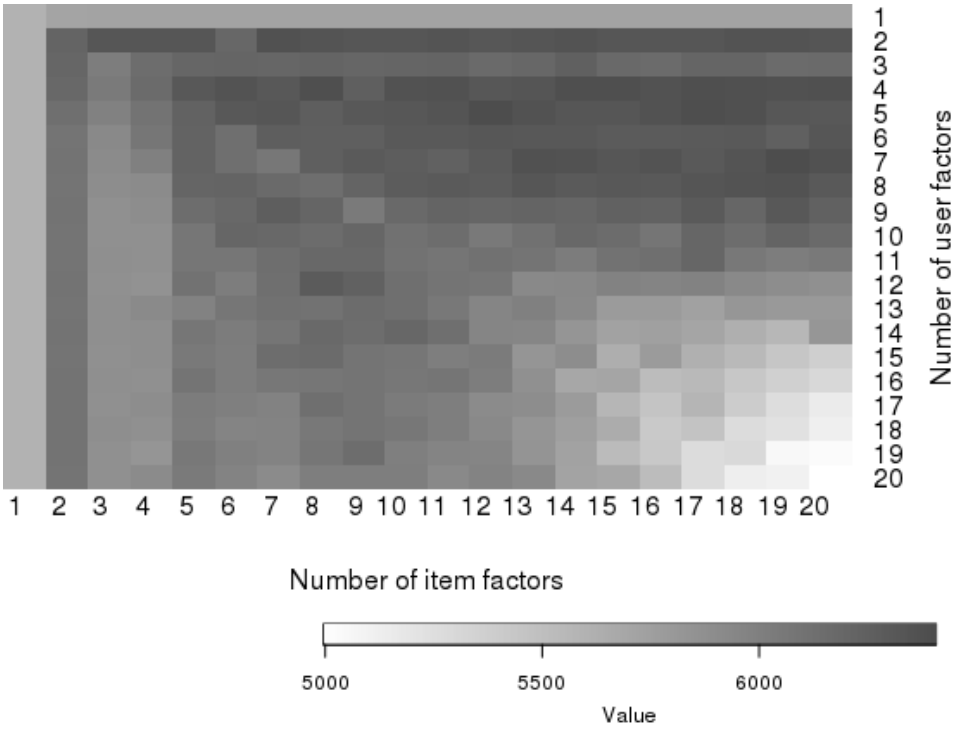


Figure B.10: Heatmap of total number of successes across all three contexts for Tucker3 decomposition with 3 item factors