Collaborative Filtering with Binary, Positive-only Data

Proefschrift

voorgelegd tot het behalen van de graad van
Doctor in de Wetenschappen: Informatica
aan de Universiteit Antwerpen
te verdedigen door

Koen VERSTREPEN

Promotor: prof. dr. Bart Goethals

Antwerpen, 2015

Collaborative Filtering with Binary, Positive-only Data
Nederlandse titel: *Collaborative Filtering met Binaire, Positieve Data*

Acknowledgements

Arriving at this point, defending my doctoral thesis, was not obvious. Not for myself, but even more so for the other people involved.

First of all Kim, my wife. She must have thought that I lost my mind, giving up my old career to start a PhD in computer science. Nevertheless, she trusted me and supported me fully. Furthermore, she did not complain once about my multiple trips to conferences, which always implied that she had to take care of our son on her own for a week. I am infinitely grateful to her.

Also Bart, my advisor, took a leap of faith when he hired me. I was not one of his summa cum laude master students destined to become an academic star. Instead, I was an unknown engineer that had left university already four years before. Fortunately, I somehow managed to convince him we would arrive at this point rather soon than late. It was no obvious decision for him, and I am grateful for the opportunity he gave me.

Starting a PhD was one of the best decisions in my life. I enjoyed every minute of it. Not the least because of the wonderful colleagues I had throughout the years. Thank you all. Special mention goes to Boris, who was the perfect office-mate: almost invisible when I wanted to focus, available when I needed him.

I would also like to thank my parents for their support and encouragement. It helps a lot to known they have my back, whatever happens.

Also a word of thanks to the members of my doctoral jury: Harald Steck, Arjen P. de Vries, Luc Martens, Floris Geerts, Kris Laukens and Chris Blondia. Their comments significantly improved this thesis.

Last but not least, thank you Jules. Because of you, I am able to put things in perspective.

Thanks!

Koen Verstrepen
Antwerpen, 2015

Contents

Abbreviations

ALS alternating least squares

AMAN all missing are negative

AMAU all missing are unknown

AUC area under the receiver operating characteristic curve

DAMIB disambiguating item-based

DCG discounted cumulative gain

EM expectation maximization

GD gradient descent

HR hit rate

IB item based (nearest neighbors)

LDA latent Dirichlet allocation

LIB length-adjusted item-based

LSA latent semantic analysis

KNN k-nearest neighbors

KUNN KUNN unified nearest neighbors

MAP mean average precision

MAR missing at random

ML maximum likelihood

MNAR missing not at random

MRR mean reciprocial rank

nDCG normalized discounted cumulative gain

pLSA probabilistic latent semantic analysis

ROC receiver operating characteristic

SGD stochastic gradient descent

SVD singular value decomposition

UB user based (nearest neighbors)

VI variational inference

CHAPTER 1

Introduction

Increasingly, people are overwhelmed by an abundance of choice. Via the World Wide Web, everybody has access to a wide variety of news, opinions, (encyclopedic) information, social contacts, books, music, videos, pictures, products, holiday accommodations, jobs, houses, and many other *items*; from all over the world. However, from the perspective of a particular person, the vast majority of items is irrelevant; and the few relevant items are difficult to find because they are buried under a large pile or irrelevant ones. There exist, for example, lots of books that I would enjoy reading, if only I could identify them. Moreover, not only do people fail to find relevant existing items, niche items fail to be created because it is anticipated that the target audience will not be able to find them under the pile of irrelevant items. Certain books, for example, are never written because writers anticipate they will not be able to reach a sufficiently large portion of their target audience, although the audience exists. *Recommender systems* contribute to overcome these difficulties by *connecting* individuals with items relevant to them. A good book recommender system, for example, would typically recommend me 3 books that I would enjoy reading, that I did not know yet, that are sufficiently different from each other, and that are suitable to read during my holiday next week. Studying recommender systems specifically, and the connection between individuals and relevant items in general, is the subject of *recommendation* research. But the relevance of recommendation research goes beyond connecting users with items. Recommender systems can, for example, also connect genes with diseases, biological targets with drug compounds, words with documents, tags with photos, etc.

1.1 Collaborative Filtering

Collaborative filtering is a principal problem in recommendation research. In the most abstract sense, collaborative filtering is the problem of weighting missing edges in a bipartite graph.

The concrete version of this problem that got most attention until recently is *rating prediction*. In rating prediction, one set of nodes in the bipartite graph represent *users*, the other set of nodes represent items, an edge with weight r between user u and item i expresses that u has given i the rating r, and the task is to predict the missing ratings. Recently, the attention for rating prediction diminished because of multiple reasons. First, collecting rating data is relatively expensive in the sense that it requires a non negligible effort from the users. Second, user ratings do not correlate as well with user behavior as one would expect. Users tend to give high ratings to items they think they should consume, for example a famous book by Dostoyevsky. However, they rather read Superman comic books, which they rated much lower. Finally, in many applications, predicting ratings is not the final goal, and the predicted ratings are only used to find the most relevant items for every user. Consequently, high ratings need to be accurate whereas the exact value of low ratings is irrelevant. However, in rating prediction high and low ratings are equally important.

Today, attention is increasingly shifting towards collaborative filtering with *binary, positive-only data*. In this version, edges are unweighted, an edge between user u and item i expresses that user u has given positive feedback about item i, and the task is to attach to every missing edge between a user u and an item i a score that indicates the suitability of recommending i to u. Binary, positive-only data is typically associated with implicit feedback such as items bought, videos watched, songs listened to, books checked out from a library, adds clicked on, etc. However, it can also be the result of explicit feedback, such as *likes* on social networking sites. Collaborative filtering with binary, positive-only data is the subject of this thesis. To enhance the readability, we sometimes omit the specification 'binary, positive-only' and use the abbreviated term 'collaborative filtering'.

Besides the bipartite graph, five types of extra information can be available. First, there can be item content or item metadata. In the case of books, for example, the content is the full text of the book and the metadata can include the writer, the publisher, the year it was published etc. Methods that exclusively use this kind of information are typically classified as *content based*. Methods that combine this kind of information with a collaborative filtering method are typically classified as *hybrid*. Second, there can be user metadata such as gender, age, location, etc. Third, users can be connected with each other in an extra, unipartite graph. A typical example is a social network between the users. An analogous graph can exist for the items. Finally, there can be contextual information such as location, date, time, intent, company, device, etc. Exploiting information besides the bipartite graph, is out of the scope of this thesis.

1.2 Relation to Other Domains

To emphasize the unique aspects of collaborative filtering, we highlight the commonalities and differences with two related data science problems: classification and association rule mining.

First, collaborative filtering is equivalent to jointly solving many one-class classification problems, in which every one-class classification problem corresponds to one of the items. In the classification problem that corresponds to item i, i serves as the class, all other items serve as the features, the users that have i as a known preference serve as labeled examples and the other users serve as unlabeled examples. Amazon.com, for example, has more than 200 million items in its catalog, hence solving the collaborative filtering problem for Amazon.com is equivalent to jointly solving more than 200 million one-class classification problems, which obviously requires a distinct approach. However, collaborative filtering is more than efficiently solving many one-class classification problems. Because they are tightly linked, *jointly* solving all classification problems allows for sharing information between them. The individual classification problems share most of their features; and while i serves as the class in one of the classification problems, it serves as a feature in all other classification problems.

Second, association rule mining also assumes bipartite, unweighted data and can therefore be applied to datasets used for collaborative filtering. Furthermore, recommending item i to user u can be considered as the application of the association rule $I(u) \rightarrow i$, with $I(u)$ the itemset containing the known preferences of u. However, the goals of association rule mining and collaborative filtering are different. If a rule $I(u) \rightarrow i$ is crucial for recommending i to u, but irrelevant on the rest of the data, giving the rule a high score is desirable for collaborative filtering, but typically not for association rule mining.

1.3 Preliminaries

We introduced collaborative filtering as the problem of weighting missing edges in a bipartite graph. Typically, however, this bipartite graph is represented by its adjacency matrix, which is called the preference matrix.

Let $\mathcal{U}$ be a set of users and $\mathcal{I}$ a set of items. We are given a preference matrix with training data $\mathbf{R} \in \{0,1\}^{|\mathcal{U}|\times|\mathcal{I}|}$. $\mathbf{R}_{ui} = 1$ indicates that there is a known preference of user $u \in \mathcal{U}$ for item $i \in \mathcal{I}$. $\mathbf{R}_{ui} = 0$ indicates that there is no such information. Notice that the absence of information means that either there exists no preference or there exists a preference but it is not known.

Collaborative filtering methods compute for every user-item-pair (u,i) a recommendation score $s(u,i)$ that indicates the suitability of recommending i to u. Typically, the user-item-pairs are (partially) sorted by their recommendation scores. We define the matrix $\mathbf{S} \in \mathbb{R}^{|\mathcal{U}|\times|\mathcal{I}|}$ as $\mathbf{S}_{ui} = s(u,i)$.

Furthermore, $c(x)$ gives the count of x, meaning

$$c(x) = \begin{cases} \sum_{i\in\mathcal{I}} R_{xi} & \text{if } x \in \mathcal{U} \\ \sum_{u\in\mathcal{U}} R_{ux} & \text{if } x \in \mathcal{I}. \end{cases}$$

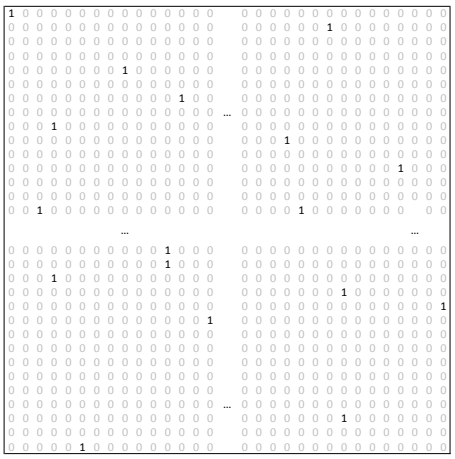

Figure 1.1: Typical sparsity of training data matrix $\mathbf{R}$ for binary, positive-only collaborative filtering.

Although we conveniently call the elements of $\mathcal{U}$ *users* and the elements of $\mathcal{I}$ *items*, these sets can contain any type of object. In the case of online social networks, for example, both sets contain the people that participate in the social network, i.e. $\mathcal{U} = \mathcal{I}$, and $\mathbf{R}_{ui} = 1$ if there exists a friendship link between person u and person i. In image tagging/annotation problems, $\mathcal{U}$ contains images, $\mathcal{I}$ contains words, and $\mathbf{R}_{ui} = 1$ if image u was tagged with word i. In chemogenomics, an early stage in the drug discovery process, $\mathcal{U}$ contains active drug compounds, $\mathcal{I}$ contains biological targets, and $\mathbf{R}_{ui} = 1$ if there is a strong interaction between compound u and biological target i.

Typical datasets for collaborative filtering are extremely sparse, which makes it a challenging problem. The sparsity $\mathcal{S}$, computed as

$$\mathcal{S} = 1 - \frac{\sum_{(u,i)\in\mathcal{U}\times\mathcal{I}} \mathbf{R}_{ui}}{|\mathcal{U}| \cdot |\mathcal{I}|}, \tag{1.1}$$

typically ranges from 0.98 to 0.999 and is visualized in Figure 1.1. This means that a score must be computed for approximately 99% of the (u,i)-pairs based on only 1% of the (u,i)-pairs. This is undeniably a challenging task.

1.4 Thesis Outline

The remainder of this thesis is organized as follows:

- In **Chapter 2**, we survey the state of the art concerning collaborative filtering with binary, positive-only data. The backbone of our survey is an innovative, unified matrix factorization perspective on collaborative filtering methods, also those that are typically not associated with matrix factorization models, such as nearest neighbors methods and association rule mining-based methods. From this perspective, a collaborative filtering algorithm consists

of three building blocks: a matrix factorization model, a deviation function and a numerical minimization algorithm. By comparing methods along these three dimensions, we were able to highlight surprising commonalities and key differences.

- In **Chapter 3**, we introduce a reformulation that unifies user- and item-based nearest neighbors methods. We use this reformulation to propose a novel method that incorporates the best of both worlds, and outperforms state-of-the-art methods. Additionally, we propose a method for naturally explaining the recommendations made by our algorithm and show that this method is also applicable to existing user-based nearest neighbors methods, which were believed to have no natural explanation.

- In **Chapter 4**, we consider the setting in which multiple users share a single account. A typical example is a single shopping account for the whole family. Traditional recommender systems fail in this situation. If contextual information is available, context aware recommender systems are the state-of-the-art solution. Yet, often no contextual information is available. Therefore, we introduce the challenge of recommending to shared accounts in the absence of contextual information. We propose a solution to this challenge for all cases in which the reference recommender system is an item-based collaborative filtering recommender system. We experimentally show the advantages of our proposed solution for tackling the problems that arise from the existence of shared accounts on multiple datasets.

- In **Chapter 5**, we consider the efficient computation of the neighborhoods for neighborhood-based collaborative filtering methods. More generally, we are given a large collection of sparse vector data in a high dimensional space; and we investigate the problem of efficiently computing the k most similar vectors to every vector. Two common names for this task are computing the k-nearest neighbors graph and performing a k-nearest neighbors self-join. Currently, exact, efficient algorithms exist for low dimensional data. For high dimensional data, on the other hand, only approximate, efficient algorithms have been proposed. However, the existing approaches do not discriminate between dense and sparse data. By focusing on sparse data, we are able to propose two inverted index-based algorithms that exploit the sparseness in the data to more efficiently compute the exact k-nearest neighbors graph for high dimensional data. We first propose a simple algorithm based on dynamic indexing. Afterwards, we propose a second algorithm that extends the first one by introducing a virtual threshold that enables the exploitation of various upper bounds for pruning candidate neighbors. We experimentally show that our approaches result into significant speedups over state-of-the-art approaches.

- In **Chapter 6**, we summarize the main contributions of this thesis and give an outlook on future work.

CHAPTER 2

A Survey of Collaborative Filtering Methods for Binary, Positive-only Data

Existing surveys on collaborative filtering assume the availability of explicit ratings of users for items. However, in many cases these ratings are not available and only binary, positive-only data is available. Binary, positive-only data is typically associated with implicit feedback such as items bought, videos watched, ads clicked on, etc. However, it can also be the results of explicit feedback such as likes on social networking sites. Because binary, positive-only data contains no negative information, it needs to be treated differently than rating data. As a result of the growing relevance of this problem setting, the number of publications in this field increases rapidly. In this chapter[1], we provide an overview of the existing work from an innovative perspective that allows us to emphasize surprising commonalities and key differences.

[1] This chapter is based on work under submission as "A Survey of Collaborative Filtering Methods for Binary, Positive-only Data" by Koen Verstrepen, Kanishka Bhaduri and Bart Goethals [101].

2.1 Framework

In the most general sense, every method for collaborative filtering is defined as a function $\mathcal{F}$ that computes the recommendation scores $\mathbf{S}$ based on the data $\mathbf{R}$:

$$\mathbf{S} = \mathcal{F}(\mathbf{R}).$$

Since different methods $\mathcal{F}$ originate from different intuitions about the problem, they are typically explained from very different perspectives. In the literature on recommender systems in general and collaborative filtering specifically, two dominant perspectives have emerged: the model based perspective and the memory-based perspective. Unfortunately, these two are often described as two fundamentally separate classes of methods, instead of merely two perspectives on the same class of methods [77, 41]. As a result, the comparison between methods often remains superficial.

We, however, will explain many different collaborative filtering methods $\mathcal{F}$ from one and the same perspective. As such we facilitate the comparison and classification of these methods. Our perspective is a matrix factorization framework in which every method $\mathcal{F}$ consists of three fundamental building blocks: a factorization model of the recommendation scores $\mathbf{S}$, a deviation function that measures the deviation between the data $\mathbf{R}$ and the recommendation scores $\mathbf{S}$, and a minimization procedure that tries to find the model parameters that minimize the deviation function.

First, the **factorization model** computes the matrix of recommendation scores $\mathbf{S}$ as a link function l of a sum of T terms in which a term t is the product of F_t factor matrices:

$$\mathbf{S} = l\left(\sum_{t=1}^{T}\prod_{f=1}^{F_t}\mathbf{S}^{(t,f)}\right). \tag{2.1}$$

For many methods l is the identity function, $T = 1$ and $F_1 = 2$. In this case, the factorization is given by:

$$\mathbf{S} = \mathbf{S}^{(1,1)}\mathbf{S}^{(1,2)}. \tag{2.2}$$

Because of their dimensions, $\mathbf{S}^{(1,1)} \in \mathbb{R}^{|\mathcal{U}|\times D}$ and $\mathbf{S}^{(1,2)} \in \mathbb{R}^{D\times|\mathcal{I}|}$ are often called the user-factor matrix and item-factor matrix respectively. Figure 2.1 visualizes Equation 2.2. More complex models are often more realistic, but generally contain more parameters which increases the risk of overfitting.

Second, the number of terms T, the number of factor matrices F_t and the dimensions of the factor matrices are an integral part of the model, independent of the data $\mathbf{R}$[2]. Every entry in the factor matrices however, is a parameter that needs to be computed based on the data $\mathbf{R}$. We collectively denote these parameters as θ. Whenever we want to emphasize that the matrix of recommendation scores $\mathbf{S}$ is dependent on these parameters, we will write it as $\mathbf{S}(\theta)$. The model in Figure 2.1, for example, contains $(|\mathcal{U}| + |\mathcal{I}|)\cdot D$ parameters. Computing all the parameters in a factorization model is done by minimizing the *deviation* between the data $\mathbf{R}$ and the parameters θ. This deviation is measured by the **deviation function** $\mathcal{D}(\theta, \mathbf{R})$.

[2] High level statistics of the data might be taken into consideration to choose the model. There is however no clear, direct dependence on the data.

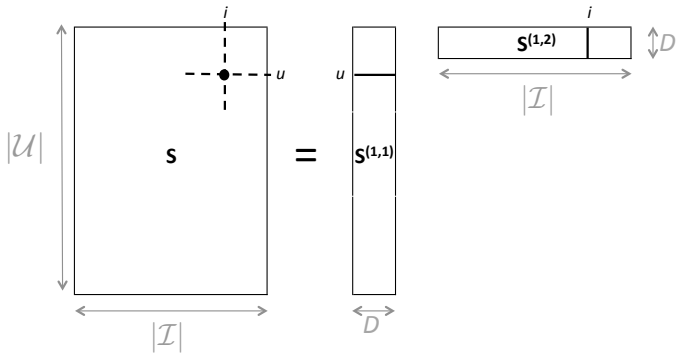

Figure 2.1: Matrix Factorization with 2 factor matrices (Eq. 2.2).

Formally we compute the *optimal* values θ^* of the parameters θ as

$$\theta^* = \arg\min_{\theta} \mathcal{D}(\theta, \mathbf{R}).$$

Many deviation functions exist, and every deviation function mathematically expresses a different interpretation of the concept *deviation*.

Third, efficiently computing the parameters that minimize a deviation function is often non trivial because the majority of deviation functions is non-convex in the parameters of the factorization model. In that case, minimization algorithms can only compute parameters that correspond to a local minimum. The initialization of the parameters in the factorization model and the chosen hyperparameters of the minimization algorithm determine which local minimum will be found. If the value of the deviation function in this local minimum is not much higher than that of the global minimum, it is considered a good minimum. An intuitively appealing deviation function is worthless if there exists no algorithm that can efficiently compute parameters that correspond to a good local minimum of this deviation function.

Finally, we assume a basic usage scenario in which model parameters are recomputed periodically. Typically, a few times every 24 hours. Computing the recommendation scores for a user based on the model, and extracting the items corresponding to the top-N scores, is assumed to be performed in *real time*. Specific aspects of scenarios in which models need to be recomputed in real time, are out of the scope of this survey.

In this chapter, we first survey existing models in Section 2.2. Next, we compare the existing deviation functions in Section 2.3. Afterwards, we discuss the minimization algorithms that are used for fitting the model parameters to the deviation functions in Section 2.4. Finally, we discuss the applicability of rating-based methods in Section 2.5 and conclude in Section 2.6.

2.2 Factorization Models

Equation 2.1 gives a general description of all models for collaborative filtering. In this section, we discuss how the specific collaborative filtering models map to this equation.

2.2.1 Basic Models

A statistically well founded method is probabilistic latent semantic analysis (pLSA) by Hofmann, which is centered around the so called aspect model [36]. Hofmann models the probability $p(i|u)$ that a user u will prefer an item i as the mixture of D probability distributions induced by the hidden variables d:

$$p(i|u) = \sum_{d=1}^{D} p(i,d|u).$$

Furthermore, by assuming u and i conditionally independent given d, he obtains:

$$p(i|u) = \sum_{d=1}^{D} p(i|d) \cdot p(d|u).$$

This model corresponds to a basic two-factor matrix factorization:

$$\begin{aligned} \mathbf{S} &= \mathbf{S}^{(1,1)}\mathbf{S}^{(1,2)} \\ \mathbf{S}_{ui} &= p(i|u) \\ \mathbf{S}^{(1,1)}_{ud} &= p(d|u) \\ \mathbf{S}^{(1,2)}_{di} &= p(i|d), \end{aligned} \tag{2.3}$$

in which the $|\mathcal{U}| \times D$ parameters in $\mathbf{S}^{(1,1)}_{ud}$ and the $D \times |\mathcal{I}|$ parameters in $\mathbf{S}^{(1,2)}_{di}$ are a priori unknown and need to be computed based on the data $\mathbf{R}$. An appealing property of this model is the probabilistic interpretation of both the parameters and the recommendation scores. Fully in line with the probabilistic foundation, the parameters are constrained as:

$$\begin{aligned} \mathbf{S}^{(1,1)}_{ud} &\geq 0 \\ \mathbf{S}^{(1,2)}_{di} &\geq 0 \\ \sum_{i \in \mathcal{I}} p(i|d) &= 1 \\ \sum_{d=1}^{D} p(d|u) &= 1, \end{aligned} \tag{2.4}$$

expressing that both factor matrices are non-negative and that all row sums of $\mathbf{S}^{(1,1)}$ and $\mathbf{S}^{(1,2)}$ must be equal to 1 since they represent probabilities.

Latent dirichlet allocation (LDA) is a more rigorous statistical model, which puts Dirichlet priors on the parameters $p(d|u)$ [12, 36]. However, for collaborative filtering these priors are integrated out and the resulting model for computing recommendation scores is again a simple two factor factorization model.

The aspect model also has a geometric interpretation. In the training data $\mathbf{R}$, every user is profiled as a binary vector in a $|\mathcal{I}|$-dimensional space in which every dimension corresponds to an item. Analogously, every item is profiled as a binary vector in an $|\mathcal{U}|$-dimensional space in which every dimension corresponds to a user. Now, in the factorized representation, every hidden variable d represents a dimension of a D-dimensional space. Therefore, the matrix factorization $\mathbf{S}$ =

$\mathbf{S}^{(1,1)}\mathbf{S}^{(1,2)}$ implies a transformation of both user- and item-vectors to the same D-dimensional space. A row vector $\mathbf{S}^{(1,1)}_{u\cdot} \in \mathbb{R}^{1\times D}$ is the representation of user u in this D-dimensional space and a column vector $\mathbf{S}^{(1,2)}_{\cdot i} \in \mathbb{R}^{D\times 1}$ is the representation of item i in this D-dimensional space. Figure 2.1 visualizes the user-vector of user u and the item-vector of item i. Now, as a result of the factorization model, a recommendation score $\mathbf{S}_{ui}$ is computed as the dotproduct of the user-vector of u with the item-vector of i:

$$\begin{aligned}\mathbf{S}^{(1,1)}_{u\cdot} \cdot \mathbf{S}^{(1,2)}_{\cdot i} &= \sum_{d=1}^{D} \mathbf{S}^{(1,1)}_{ud}\mathbf{S}^{(1,2)}_{di} \\ &= ||\mathbf{S}^{(1,1)}_{u\cdot}|| \cdot ||\mathbf{S}^{(1,2)}_{\cdot i}|| \cdot \cos(\phi_{ui}) \\ &= ||\mathbf{S}^{(1,2)}_{\cdot i}|| \cdot \cos(\phi_{ui}),\end{aligned} \tag{2.5}$$

with ϕ_{ui} the angle between both vectors and $||\mathbf{S}^{(1,1)}_{u\cdot}|| = 1$ a probabilistic constraint of the model (Eq.2.4). Therefore, an item i will be recommended if its vector norm $||\mathbf{S}^{(1,2)}_{\cdot i}||$ is large and ϕ_{ui}, the angle of its vector with the user vector, is small. From this geometric interpretation we learn that the recommendation scores computed with the model in Equation 2.3 contain both a personalized factor $\cos(\phi_{ui})$ and a non-personalized, popularity based factor $||\mathbf{S}^{(1,2)}_{\cdot i}||$.

Many other authors adopted this two-factor model, however they abandoned its probabilistic foundation by removing the constraints on the parameters (Eq.2.4) [38, 64, 65, 85, 114, 75, 83, 93, 29, 21]:

$$\mathbf{S} = \mathbf{S}^{(1,1)}\mathbf{S}^{(1,2)}. \tag{2.6}$$

Yet another interpretation of this two-factor model is that every hidden variable d represents a cluster containing both users and items. A large value $\mathbf{S}^{(1,1)}_{ud}$ means that users u has a large degree of membership in cluster d. Similarly, a large value $\mathbf{S}^{(1,2)}_{di}$ means that item i has a large degree of membership in clusters d. As such, pLSA can be interpreted as a soft clustering model. According to the interpretation, an item i will be recommended to a user u if they have high degrees of membership in the same clusters.

Although much less common, also hard clustering models for collaborative filtering exist. Hofmann and Puzicha [37] proposed the model

$$p(i|u, e(u) = c, d(i) = d) = p(i)\phi(e, d), \tag{2.7}$$

in which $e(u)$ indicates the cluster user u belongs to and $d(i)$ indicates the cluster item i belongs to. Furthermore, $\phi(e,d)$ is the association value between the user-cluster e and the item-cluster d. This cluster association factor increases or decreases the probability that u likes i relative to the independence model $p(i|u) = p(i)$. As opposed to the aspect model, this model assumes E user-clusters only containing users and D item-clusters only containing items. Furthermore a user or item belongs to exactly one cluster.

The factorization model corresponding to this approach is:

$$\begin{aligned}
&\mathbf{S} = \mathbf{S}^{(1,1)}\mathbf{S}^{(1,2)}\mathbf{S}^{(1,3)}\mathbf{S}^{(1,4)},\\
&\mathbf{S}_{ui} = p(i|u),\\
&\mathbf{S}^{(1,1)}_{ue} = \mathbb{I}(e(u) = e),\\
&\mathbf{S}^{(1,2)}_{ed} = \phi(e,d),\\
&\mathbf{S}^{(1,3)}_{di} = \mathbb{I}(d(i) = d),\\
&\mathbf{S}^{(1,4)}_{ij} = p(i)\cdot\mathbb{I}(i = j),
\end{aligned}$$

in which $\mathbb{I}(\mathit{true}) = 1$, $\mathbb{I}(\mathit{false}) = 0$, and E and D are hyperparameters. The $|\mathcal{U}| \times E$ parameters in $\mathbf{S}^{(1,1)}$, $E \times D$ parameters in $\mathbf{S}^{(1,2)}$, $D \times |\mathcal{I}|$ parameters in $\mathbf{S}^{(1,3)}$ and the $|\mathcal{I}|$ parameters $\mathbf{S}^{(1,4)}$ need to be computed based on the data. Ungar and Foster [99] proposed a similar hard clustering method.

2.2.2 Explicitly Biased Models

In the above models, the recommendation score $\mathbf{S}_{ui}$ of an item i for a user u is the product of a personalized factor with an item-bias factor related to item-popularity. In Equation 2.5 the personalized factor is $\cos(\phi_{ui})$, and the bias factor is $||\mathbf{S}^{(1,2)}_{\cdot i}||$. In Equation 2.7 the personalized factor is $\phi(e,d)$, and the bias factor is $p(i)$. Other authors [47, 68, 42] proposed to model item-, and user-biases, as explicit terms instead of implicit factors. This results into the following factorization model:

$$\begin{aligned}
&\mathbf{S} = \sigma\left(\mathbf{S}^{(1,1)} + \mathbf{S}^{(2,1)} + \mathbf{S}^{(3,1)}\mathbf{S}^{(3,2)}\right)\\
&\mathbf{S}^{(1,1)}_{ui} = b_u\\
&\mathbf{S}^{(2,1)}_{ui} = b_i,
\end{aligned} \tag{2.8}$$

with σ the sigmoid link-function, and $\mathbf{S}^{(1,1)}, \mathbf{S}^{(2,1)} \in \mathbb{R}^{|\mathcal{U}|\times|\mathcal{I}|}$ the user- and item-bias matrices in which all columns of $\mathbf{S}^{(1,1)}$ are identical and also all rows of $\mathbf{S}^{(2,1)}$ are identical. The $|\mathcal{U}|$ parameters in $\mathbf{S}^{(1,1)}$, $|\mathcal{I}|$ parameters in $\mathbf{S}^{(2,1)}$, $|\mathcal{U}|\cdot D$ parameters in $\mathbf{S}^{(3,1)}$, and $|\mathcal{I}|\cdot D$ parameters in $\mathbf{S}^{(3,2)}$ need to be computed based on the data. D is a hyperparameter of the model. The goal of explicitly modeling the bias terms is to make the interaction term $\mathbf{S}^{(3,1)}\mathbf{S}^{(3,2)}$ a pure personalization term. Although bias terms are commonly used for collaborative filtering with rating data, only a few works with collaborative filtering with binary, positive-only data use them.

2.2.3 Basic Neighborhood Models

Multiple authors proposed special cases of the basic two-factor factorization in Equation 2.6.

Item-based

In a first special case, $\mathbf{S}^{(1,1)} = \mathbf{R}$ [19, 84, 75, 1, 60]. In this case, the factorization model is given by

$$\mathbf{S} = \mathbf{R}\mathbf{S}^{(1,2)}. \tag{2.9}$$

Consequently, a user u is profiled by an $|\mathcal{I}|$-dimensional binary vector $\mathbf{R}_{u\cdot}$ and an item is profiled by an $|\mathcal{U}|$-dimensional real valued vector $\mathbf{S}^{(1,2)}_{\cdot i}$. This model is often interpreted as an *item-based neighborhood model* because the recommendation score $\mathbf{S}_{ui}$ of item i for user u is computed as

$$\begin{aligned}\mathbf{S}_{ui} &= \mathbf{R}_{u\cdot} \cdot \mathbf{S}^{(1,2)}_{\cdot i} \\ &= \sum_{j \in \mathcal{I}} \mathbf{R}_{uj} \cdot \mathbf{S}^{(1,2)}_{ji} \\ &= \sum_{j \in I(u)} \mathbf{S}^{(1,2)}_{ji},\end{aligned}$$

with $I(u)$ the known preferences of user u, and a parameter $\mathbf{S}^{(1,2)}_{ji}$ typically interpreted as the similarity between items j and i, i.e. $\mathbf{S}^{(1,2)}_{ji} = sim(j,i)$. Consequently, $\mathbf{S}^{(1,2)}$ is often called the item-similarity matrix. The SLIM method [60] adopts this model and additionally imposes the constraints

$$\begin{aligned}\mathbf{S}^{(1,2)}_{ji} &\geq 0, \\ \mathbf{S}^{(1,2)}_{ii} &= 0.\end{aligned} \tag{2.10}$$

The non-negativity is imposed to enhance interpretability. The zero-diagonal is imposed to avoid finding a trivial solution for the parameters in which every item is maximally similar to itself and has zero similarity with any other item.

This model is rooted in the intuition that good recommendations are similar to the known preferences of the user. Although they do not present it as such, Gori and Pucci [31] proposed ItemRank, a method that is based on an interesting extension of this intuition: good recommendations are similar to other good recommendations, with a bias towards the known preferences of the user. The factorization model corresponding to ItemRank is based on PageRank [62] and given by:

$$\mathbf{S} = \alpha \cdot \mathbf{S}\mathbf{S}^{(1,2)} + (1-\alpha) \cdot \mathbf{R}. \tag{2.11}$$

Because of the recursive nature of this model, $\mathbf{S}$ needs to be computed iteratively [31, 62].

User-based

Other authors proposed the symmetric counterpart of Equation 2.9 in which $\mathbf{S}^{(1,2)} = \mathbf{R}$ [79, 1, 2]. In this case, the factorization model is given by

$$\mathbf{S} = \mathbf{S}^{(1,1)}\mathbf{R}, \tag{2.12}$$

which is often interpreted as a *user-based neighborhood model* because the recommendation score $\mathbf{S}_{ui}$ of item i for user u is computed as

$$\begin{aligned}\mathbf{S}_{ui} &= \mathbf{S}^{(1,1)}_{u\cdot} \cdot \mathbf{R}_{\cdot i} \\ &= \sum_{v \in \mathcal{U}} \mathbf{S}^{(1,1)}_{uv} \cdot \mathbf{R}_{vi},\end{aligned}$$

in which a parameter $\mathbf{S}^{(1,1)}_{uv}$ is typically interpreted as the similarity between users u and v, i.e. $\mathbf{S}^{(1,1)}_{uv} = sim(u,v)$. Consequently, $\mathbf{S}^{(1,1)}$ is often called the user-similarity matrix. This model is rooted in the intuition that good recommendations are preferred by similar users. Furthermore, Aiolli [1, 2] foresees the possibility to rescale the scores such that popular items get less importance. This changes the factorization model to

$$\begin{aligned}\mathbf{S} &= \mathbf{S}^{(1,1)}\mathbf{R}\mathbf{S}^{(1,3)} \\ \mathbf{S}^{(1,3)}_{ji} &= \mathbb{I}(j=i)\cdot c(i)^{-(1-\beta)},\end{aligned}$$

in which $\mathbf{S}^{(1,3)}$ is a diagonal rescaling matrix that rescales the item scores according to the item popularity $c(i)$, and $\beta \in [0,1]$ a hyperparameter. Additionally, Aiolli [2] imposes the constraint that $\mathbf{S}^{(1,1)}$ needs to be row normalized:

$$||\mathbf{S}^{(1,1)}_{u.}|| = 1.$$

To the best of our knowledge, there exists no user-based counterpart for the ItemRank model of Equation 2.11.

Explicitly Biased

Furthermore, it is, obviously, possible to add explicit bias terms to neighborhood models. Wang et al. [106] proposed an item-based neighborhood model with one bias term:

$$\begin{aligned}\mathbf{S} &= \mathbf{S}^{(1,1)} + \mathbf{R}\mathbf{S}^{(2,2)}, \\ \mathbf{S}^{(1,1)}_{ui} &= b_i,\end{aligned}$$

and an analogous user-based model:

$$\begin{aligned}\mathbf{S} &= \mathbf{S}^{(1,1)} + \mathbf{S}^{(2,1)}\mathbf{R}, \\ \mathbf{S}^{(1,1)}_{ui} &= b_i.\end{aligned}$$

Unified

Moreover, Verstrepen and Goethals [102] showed that the item- and user-based neighborhood models are two incomplete instances of the same general neighborhood model. Consequently they propose *KUNN*, a complete instance of this general neighborhood model. The model they propose can be written as a weighted sum of a user- and an item-based models, in which the weights depend on the user u and the item i for which a recommendation score is computed:

$$\begin{aligned}\mathbf{S} &= \left(\mathbf{S}^{(1,1)}\mathbf{R}\right)\mathbf{S}^{(1,3)} + \mathbf{S}^{(2,1)}\left(\mathbf{R}\mathbf{S}^{(2,3)}\right) \\ \mathbf{S}^{(1,3)}_{ij} &= \mathbb{I}(i=j)\cdot c(i)^{-1/2} \\ \mathbf{S}^{(2,1)}_{uv} &= \mathbb{I}(u=v)\cdot c(u)^{-1/2}.\end{aligned} \tag{2.13}$$

In Chapter 3, we discuss this work in detail.

Finally, notice that the above matrix factorization based descriptions of nearest neighbors methods imply that matrix factorization methods and neighborhood methods are not two separate approaches, but two perspectives on the same approach.

2.2.4 Factored Similarity Neighborhood Models

The item-similarity matrix $\mathbf{S}^{(1,2)}$ in Equation 2.9 contains $|\mathcal{I}|^2$ parameters, which is in practice often a very large number. Consequently, it can happen that the training data is not sufficient to accurately compute this many parameters. Furthermore, one often precomputes the item-similarity matrix $\mathbf{S}^{(1,2)}$ and performs the dotproducts $\mathbf{R}_{u\cdot} \cdot \mathbf{S}^{(1,2)}_{\cdot i}$ to compute the recommendation scores $\mathbf{S}_{ui}$ in real time. In this case, the dotproduct is between two $|\mathcal{I}|$-dimensional vectors, which is often prohibitive in real time, high traffic applications. One solution[3] is to factorize the similarity matrix, which leads to the following factorization model:

$$\mathbf{S} = \mathbf{R}\mathbf{S}^{(1,2)}\mathbf{S}^{(1,2)\,T},$$

in which every row of $\mathbf{S}^{(1,2)} \in \mathbb{R}^{|\mathcal{I}| \times D}$ represents a D-dimensional item-profile vector, with D a hyperparameter [110, 17]. In this case the item-similarity matrix is equal to $\mathbf{S}^{(1,2)}\mathbf{S}^{(1,2)\,T}$, which means that the similarity $sim(i, j)$ between two items i and j is defined as the dotproduct of their respective profile vectors $\mathbf{S}^{(1,2)}_{i\cdot} \cdot \mathbf{S}^{(1,2)\,T}_{j\cdot}$. This model only contains $|\mathcal{I}| \cdot D$ parameters instead of $|\mathcal{I}|^2$ which is much less since typically $D \ll |\mathcal{I}|$. Furthermore, by first precomputing the item vectors $\mathbf{S}^{(1,2)}$ and then precomputing the D-dimensional user-profile vectors given by $\mathbf{R}\mathbf{S}^{(1,2)}$, the real time computation of a score $\mathbf{S}_{ui}$ encompasses a dotproduct between a D-dimensional user-profile vector and a D-dimensional item-profile vector. Since $D \ll |\mathcal{I}|$, this dotproduct is much less expensive and can be more easily performed in real time. Furthermore, there is no trivial solution for the parameters of this model, as is the case for the non factored item-similarity model in Equation 2.9. Consequently, it is never required to impose the constraints from Equation 2.10. To avoid scaling problems when fitting parameters, Weston et al. [110] augment this model with a diagonal normalization factor matrix:

$$\mathbf{S} = \mathbf{S}^{(1,1)}\mathbf{R}\mathbf{S}^{(1,3)}\mathbf{S}^{(1,3)\,T} \tag{2.14}$$

$$\mathbf{S}^{(1,1)}_{uv} = \mathbb{I}(u = v) \cdot c(u)^{-2/2}. \tag{2.15}$$

A limitation of this model is that it implies that the similarity matrix is symmetric. This might hurt the model's accuracy in certain applications such as recommending tags for images. For an image of the Eiffel tower that is already tagged *Eiffel tower*, for example, the tag *Paris* is a reasonable recommendation. However, for an image of the Louvre already tagged *Paris*, the tag *Eiffel tower* is a bad recommendation. Paterek solved this problem for rating data by representing every item by two separate D-dimensional vectors [71]. One vector represents the item if it serves as evidence for computing recommendations, the other vector represents the item if it serves as a candidate recommendation. In this way, they can model also asymmetric similarities. This idea is not restricted to rating data, and for binary, positive-only data, it was adopted by Steck [89]:

$$\mathbf{S} = \mathbf{S}^{(1,1)}\mathbf{R}\mathbf{S}^{(1,3)}\mathbf{S}^{(1,4)} \tag{2.16}$$

$$\mathbf{S}^{(1,1)}_{uv} = \mathbb{I}(u = v) \cdot c(u)^{-1/2}. \tag{2.17}$$

[3] Another solution is to enforce sparsity on the similarity matrix by means of the deviation function. This is discussed in Section 2.3.

Also Kabbur et al. adopted this idea. However, similar to Equation 2.8, they also add bias terms:

$$\begin{aligned}
\mathbf{S} &= \mathbf{S}^{(1,1)} + \mathbf{S}^{(2,1)} + \mathbf{S}^{(3,1)}\mathbf{R}\mathbf{S}^{(3,3)}\mathbf{S}^{(3,4)} \\
\mathbf{S}^{(1,1)}_{ui} &= b_u \\
\mathbf{S}^{(2,1)}_{ui} &= b_i \\
\mathbf{S}^{(3,1)}_{uv} &= \mathbb{I}(u=v)\cdot c(u)^{-\beta/2},
\end{aligned} \tag{2.18}$$

with $\mathbf{S}^{(3,3)} \in \mathbb{R}^{|\mathcal{I}|\times D}$ the matrix of item profiles when they serve as evidence, $\mathbf{S}^{(3,4)} \in \mathbb{R}^{D\times|\mathcal{I}|}$ the matrix of item profiles when they serve as candidates , and $\beta \in [0,1]$ and D hyperparameters.

2.2.5 Higher Order Neighborhood Models

The nearest neighbors methods discussed up till this point only consider pairwise interactions $sim(j,i)$ and/or $sim(u,v)$ and aggregate all the relevant ones for computing recommendations. Several authors [19, 56, 100, 81, 53, 58, 15] have proposed to incorporate also higher order interactions $sim(J,i)$ and/or $sim(u,V)$ with $J \subset \mathcal{I}$ and $V \subset \mathcal{U}$. Also in this case we can distinguish item-based approaches from user-based approaches.

Item-based

For the item-based approach, most authors [19, 56, 100, 53, 15] propose to replace the user-item matrix $\mathbf{R}$ in the pairwise model of Equation 2.9 by the user-itemset matrix $\mathbf{X}$:

$$\begin{aligned}
\mathbf{S} &= \mathbf{X}\mathbf{S}^{(1,2)} \\
\mathbf{X}_{uJ} &= \prod_{j\in J} \mathbf{R}_{uj},
\end{aligned} \tag{2.19}$$

with $\mathbf{S}^{(1,2)} \in \mathbb{R}^{D\times|\mathcal{I}|}$ the itemset-item similarity matrix and $\mathbf{X} \in \{0,1\}^{|\mathcal{U}|\times D}$ the user-itemset matrix, in which $D \leq 2^{|\mathcal{I}|}$ is the result of an itemset selection procedure.

The HOSLIM method [15] adopts this model and additionally imposes the constraints in Equation 2.10.

The case in which $D = 2^{|\mathcal{I}|}$ and $\mathbf{S}^{(1,2)}$ is dense, is intractable. Tractable methods either limit $D \ll 2^{|\mathcal{I}|}$ or impose sparsity on $\mathbf{S}^{(1,2)}$ via the deviation function. While we discuss the latter in Section 2.3.11, there are multiple ways to do the former. Deshpande et al. [19] limit the number of itemsets by limiting the size of J. Alternatively, Christakopoulou and Karypis [15] only consider itemsets J that were preferred by more than a minimal number of users. van Leeuwen and Puspitaningrum, on the other hand, limit the number of higher order itemsets by using an itemset selection algorithm based the minimal description length principle [100]. Finally, Menezes et al. claim that it is in certain applications possible to compute all higher order interactions if one computes all higher order interactions on demand instead of in advance [56]. However, delaying the computation does not reduce its exponential complexity. Only if a large portion of the users requires recommendations on a very infrequent basis, computations for these users can be spread over a very long period of time and their approach might be feasible.

An alternative model for incorporating higher order interactions between items consists of finding the best association rule for making recommendations [81, 58]. This corresponds to the matrix factorization model

$$\begin{aligned}\mathbf{S} &= \mathbf{X} \otimes \mathbf{S}^{(1,2)} \\ \mathbf{X}_{uJ} &= \prod_{j \in J} \mathbf{R}_{uj},\end{aligned}$$

with

$$(\mathbf{A} \otimes \mathbf{B})_{xy} = \max_{i=1\ldots m} \mathbf{A}_{xi}\mathbf{B}_{iy},$$

in which $\mathbf{A} \in \mathbb{R}^{n \times m}$ and $\mathbf{B} \in \mathbb{R}^{m \times k}$.

We did not find a convincing motivation for neither of both aggregation strategies. Moreover, multiple authors report that their attempt to incorporate higher order interactions heavily increased computational costs and did not significantly improve the results [19, 100].

User-based

Incorporating higher order interactions between users can be achieved be replacing the user-item matrix in Equation 2.12 by the userset-item matrix $\mathbf{Y}$ [53, 92]:

$$\begin{aligned}\mathbf{S} &= \mathbf{S}^{(1,1)}\mathbf{Y} \\ \mathbf{Y}_{Vi} &= \prod_{v \in V} \mathbf{R}_{vi},\end{aligned} \tag{2.20}$$

with $\mathbf{Y} \in \{0,1\}^{D \times |\mathcal{I}|}$ and, $\mathbf{S}^{(1,1)} \in \mathbb{R}^{|\mathcal{U}| \times D}$ the user-userset similarity matrix, in which $D \leq 2^{|\mathcal{U}|}$ is the result of a userset selection procedure [53, 92].

2.2.6 Multi-profile Models

When multiple users, e.g. members of the same family, share a single account, or when a user has multiple distinct tastes, the above matrix factorization models can be too limited because they aggregate all the distinct tastes of an account into one vector [104, 109, 45]. Therefore, Weston et al. [109] propose MaxMF, in which they model every account with multiple vectors instead of just one. Then, for every candidate recommendation, their model chooses the vector that maximizes the score of the candidate:

$$\mathbf{S}_{ui} = \max_{p=1\ldots P} \left(\mathbf{S}^{(1,1,p)}\mathbf{S}^{(1,2)}\right)_{ui}.$$

Kabbur and Karypis [45] argue that this approach worsens the performance for accounts with homogeneous taste or a low number of known preferences and therefore propose, NLMFi, an adapted version that combines a global account profile with multiple taste-specific account profiles:

$$\mathbf{S}_{ui} = \left(\mathbf{S}^{(1,1)}\mathbf{S}^{(1,2)}\right)_{ui} + \max_{p=1\ldots P} \left(\mathbf{S}^{(2,1,p)}\mathbf{S}^{(2,2)}\right)_{ui}.$$

Alternatively, they also propose NLMFs, a version in which $\mathbf{S}^{(2,2)} = \mathbf{S}^{(1,2)}$, i.e. the item profiles are shared between the global and the taste-specific terms:

$$\mathbf{S} = \left(\mathbf{S}^{(1,1)}\mathbf{S}^{(1,2)}\right)_{ui} + \max_{p=1\ldots P} \left(\mathbf{S}^{(2,1,p)}\mathbf{S}^{(1,2)}\right)_{ui}.$$

An important downside of the above two models is that P, the number of distinct account-profiles, is a hyperparameter that is the same for every account and cannot be too large (typically two or three) to avoid an explosion of the computational cost and the number of parameters. DAMIB-Cover [104], on the other hand, starts from the item-based model in Equation 2.9, and efficiently considers $P = 2^{c(u)}$ different profiles for every account u. Specifically, every profile corresponds to a subset $S^{(p)}$ of the known preferences of u, $I(u)$. This results in the factorization model

$$\mathbf{S}_{ui} = \max_{p=1\ldots 2^{c(u)}} \left(\mathbf{R}\mathbf{S}^{(1,2,p)}\mathbf{S}^{(1,3)}\right)_{ui}$$
$$\mathbf{S}^{(1,2,p)}_{jk} = \frac{\mathbb{I}(j=k)\cdot|S^{(p)}\cap\{j\}|}{|S^{(p)}|^{\beta}},$$

with $\mathbf{S}^{(1,2,p)}$ a diagonal matrix that selects and rescales the known preferences of u that correspond to the subset $S^{(p)} \subseteq I(u)$, and $\beta \in [0,1]$ a hyperparameter. In Chapter 4, we discuss this model in detail.

2.3 Deviation Functions

The factorization models described in the previous Section (Sec. 2.2) contain many parameters, i.e. the entries in the a priori unknown factor matrices, which we collectively denote as θ. These parameters need to be computed based on the training data $\mathbf{R}$. Computing all the parameters in a factorization model is done by minimizing the *deviation* between the training data $\mathbf{R}$ and the parameters θ of the factorization model. This deviation is measured by a deviation function $\mathcal{D}(\theta,\mathbf{R})$. Many deviation functions exist, and every deviation function mathematically expresses a different interpretation of the concept *deviation*.

The majority of deviation functions is non-convex in the parameters θ. Consequently, minimization algorithms can only compute parameters that correspond to a local minimum. The initialization of the parameters in the factorization model and the chosen hyperparameters of the minimization algorithm determine which local minimum will be found. If the value of the deviation function in this local minimum is not much higher than that of the global minimum, it is considered a good minimum.

2.3.1 Probabilistic Scores-based

Hofmann [36] proposes to compute the *optimal* parameters θ^* of the model as those that maximize the loglikelihood of the model, i.e. $\log p(\mathbf{R}|\theta)$, the logprobability of the known preferences given the model:

$$\theta^* = \arg\max_{\theta} \sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}} \mathbf{R}_{ui}\log p(\mathbf{R}_{ui}|\theta). \tag{2.21}$$

Furthermore, he models $p(\mathbf{R}_{ui}=1|\theta)$ with $\mathbf{S}_{ui}$, i.e. he interprets the scores $\mathbf{S}$ as probabilities. This gives

$$\theta^* = \arg\max_{\theta} \sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}} \mathbf{R}_{ui}\log\mathbf{S}_{ui},$$

which is equivalent to minimizing the deviation function

$$\mathcal{D}(\theta, \mathbf{R}) = -\sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} \mathbf{R}_{ui} \log \mathbf{S}_{ui}. \tag{2.22}$$

Recall that the parameters θ are not directly visible in the right hand side of this equation but that every score $\mathbf{S}_{ui}$ is computed based on the factorization model which contains the parameters θ, i.e. we use the shorthand notation $\mathbf{S}_{ui}$ instead of the full notation $\mathbf{S}_{ui}(\theta)$.

This deviation function was also adopted by Blei et al. in their approach called *latent dirichlet allocation (LDA)* [12].

Furthermore, notice that Hofmann only maximizes the logprobability of the observed feedback and ignores the missing preferences. This is equivalent to the assumption that there is no information in the missing preferences, which implicitly corresponds to the assumption that feedback is *missing at random (MAR)* [87]. Clearly, this is not a realistic assumption, since negative feedback is missing by definition, which is obviously non random. Moreover, the number of items in collaborative filtering problems is typically very large, and only a very small subset of them will be preferred by a user. Consequently, the probability that a missing preference is actually not preferred is high. Hence, in reality, the feedback is *missing not at random (MNAR)*, and a good deviation function needs to account for this [87].

One approach is to assume, for the purposes of defining the deviation function, that *all* missing preferences are not preferred. This assumption is called *all missing are negative (AMAN)* [65]. Under this assumption, the parameters that maximize the loglikelihood of the model are computed as

$$\theta^* = \arg\max_{\theta} \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} \log p(\mathbf{R}_{ui} | \theta).$$

For binary, positive-only data, one can model $p(\mathbf{R}_{ui}|\theta)$ as $\mathbf{S}_{ui}^{\mathbf{R}_{ui}} \cdot (1 - \mathbf{S}_{ui})^{(1-\mathbf{R}_{ui})}$. In this case, the parameters are computed as

$$\theta^* = \arg\max_{\theta} \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} \left(\mathbf{R}_{ui} \log \mathbf{S}_{ui} + (1 - \mathbf{R}_{ui}) \log(1 - \mathbf{S}_{ui}) \right).$$

While the AMAN assumption is more realistic than the MAR assumption, it adopts a conceptually flawed missing data model. Specifically, it assumes that all missing preferences are not preferred, which contradicts the goal of collaborative filtering: to find the missing preferences that are actually preferred. A better missing data model still assumes that all missing preferences are not preferred. However, it attaches a lower confidence to the assumption that a missing preference is not preferred, and a higher confidence to the assumption that an observed preference is indeed preferred. One possible way to apply this missing data model was proposed by Steck [87]. Although his original approach is more general, we give a specific simplified version that for binary, positive-only data:

$$\theta^* = \arg\max_{\theta} \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} \left(\mathbf{R}_{ui} \log \mathbf{S}_{ui} + \alpha \cdot (1 - \mathbf{R}_{ui}) \log (1 - \mathbf{S}_{ui}) \right), \tag{2.23}$$

in which the hyperparameter $\alpha < 1$ attaches a lower importance to the contributions that correspond to $\mathbf{R}_{ui} = 0$. Johnson [42] proposed a very similar computation, but

does not motivate why he deviates from the theoretically well founded version of Steck. Furthermore, Steck adds regularization terms to avoid overfitting and finally propose the deviation function

$$\mathcal{D}(\theta, \mathbf{R}) = \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} \left(-\mathbf{R}_{ui} \log \mathbf{S}_{ui} - \alpha \cdot (1 - \mathbf{R}_{ui}) \cdot \log(1 - \mathbf{S}_{ui}) + \lambda \cdot \alpha \cdot (||\theta_u||_F^2 + ||\theta_i||_F^2) \right),$$

with $||\cdot||_F$ the Frobenius norm, λ a regularization hyperparameter, and θ_u, θ_i the vectors that group the model parameters related to user u and item i respectively.

2.3.2 Basic Squared Error-based

Most deviation functions, however, abandon the interpretation that the scores $\mathbf{S}$ are probabilities. In this case, one can choose to model $p(\mathbf{R}_{ui}|\theta)$ with a normal distribution $\mathcal{N}(\mathbf{R}_{ui}|\mathbf{S}_{ui}, \sigma)$. By additionally adopting the AMAN assumption, the *optimal* parameters are computed as the ones that maximize the loglikelihood $\log p(\mathbf{R}|\theta)$:

$$\theta^* = \arg\max_{\theta} \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} \log \mathcal{N}(\mathbf{R}_{ui}|\mathbf{S}_{ui}, \sigma),$$

which is equivalent to

$$\begin{aligned} \theta^* &= \arg\max_{\theta} \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} -(\mathbf{R}_{ui} - \mathbf{S}_{ui})^2 \\ &= \arg\min_{\theta} \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} (\mathbf{R}_{ui} - \mathbf{S}_{ui})^2. \end{aligned}$$

The Eckart-Young theorem [24] states that the scores matrix $\mathbf{S}$ that results from these parameters θ^*, is the same as that found by singular value decomposition (SVD) with the same dimensions of the training data matrix $\mathbf{R}$. As such, the theorem relates the above approach of minimizing the squared error between $\mathbf{R}$ and $\mathbf{S}$ to *latent semantic analysis (LSA)* [18] and the SVD based collaborative filtering methods [17, 80].

Alternatively, it is possible to compute the *optimal* parameters as those that maximize the logposterior $\log p(\theta|\mathbf{R})$, which relates to the loglikelihood $\log p(\mathbf{R}|\theta)$ as

$$p(\theta|\mathbf{R}) \propto p(\mathbf{R}|\theta) \cdot p(\theta).$$

When $p(\theta)$, the prior distribution of the parameters, is chosen to be a zero-mean, spherical normal distribution, maximizing the logposterior is equivalent to minimizing the deviation function [57]

$$\mathcal{D}(\theta, \mathbf{R}) = \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} \left((\mathbf{R}_{ui} - \mathbf{S}_{ui})^2 + \lambda_u \cdot ||\theta_u||_F^2 + \lambda_i \cdot ||\theta_i||_F^2 \right),$$

with λ_u, λ_i regularization hyperparameters. Hence, maximizing the logposterior instead of the loglikelihood is equivalent to adding a regularization term. This deviation function is adopted by the FISM and NLMF methods [46, 45].

In an alternative interpretation of this deviation function, $\mathbf{S}$ is a factorized approximation of $\mathbf{R}$ and the deviation function minimizes the squared error of the approximation. The regularization with the Frobenius norm is added to avoid overfitting. For the SLIM and HOSLIM methods, an alternative regularization term is

proposed:

$$\mathcal{D}(\theta, \mathbf{R}) = \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} (\mathbf{R}_{ui} - \mathbf{S}_{ui})^2 + \sum_{t=1}^{T} \sum_{f=1}^{F} \lambda_R ||\mathbf{S}^{(t,f)}||_F^2 + \lambda_1 ||\mathbf{S}^{(t,f)}||_1,$$

with $||\cdot||_1$ the l_1-norm. Whereas the role of the Frobenius-norm is to avoid overfitting, the role of the l_1-norm is to introduce sparsity. The combined use of both norms is called elastic net regularization, which is known to implicitly group correlated items [60]. The sparsity induced by the l_1-norm regularization lowers the memory required for storing the model and allows to speed-up the computation of recommendations by means of the sparse dotproduct. Even more sparsity can be obtained by fixing a majority of the parameters to 0, based on a simple feature selection method. Ning and Karypis [60] empirically show that this significantly reduces runtimes, while only slightly reducing the accuracy.

2.3.3 Weighted Squared Error-based

Also these squared error based deviation functions can be adapted to diverge from the AMAN assumption to a missing data model that attaches lower confidence to the missing preferences [38, 65, 87]:

$$\mathcal{D}(\theta, \mathbf{R}) = \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} \mathbf{W}_{ui} (\mathbf{R}_{ui} - \mathbf{S}_{ui})^2 + \sum_{t=1}^{T} \sum_{f=1}^{F} \lambda_{tf} ||\mathbf{S}^{(t,f)}||_F^2, \tag{2.24}$$

in which $\mathbf{W} \in \mathbb{R}^{|\mathcal{U}| \times |\mathcal{I}|}$ assigns a weight to every value in $\mathbf{R}$. The higher $\mathbf{W}_{ui}$, the higher the confidence about $\mathbf{R}_{ui}$. Hu et al. [38] provide two potential definitions of $\mathbf{W}$:

$$\mathbf{W}_{ui} = 1 + \beta \mathbf{R}_{ui},$$
$$\mathbf{W}_{ui} = 1 + \alpha \log(1 + \mathbf{R}_{ui}/\epsilon),$$

with α, β, ϵ hyperparameters. From the above definitions, it is clear that this method is not limited to binary data, but works on positive-only data in general. We, however, only consider its use for binary, positive-only data. Equivalently, Steck [87] proposed:

$$\mathbf{W}_{ui} = \mathbf{R}_{ui} + (1 - \mathbf{R}_{ui}) \cdot \alpha,$$

with $\alpha < 1$.

Additionally, Steck [88] pointed out that a preference is more likely to show up in the training data if the item is more popular. To compensate for this bias, Steck proposes to weight the known preferences non-uniformly:

$$\mathbf{W}_{ui} = \mathbf{R}_{ui} \cdot \frac{C}{c(i)^\beta} + (1 - \mathbf{R}_{ui}) \cdot \alpha,$$

with C a constant, R the number of non-zeros in the training data $\mathbf{R}$, and $\beta \in [0, 1]$ a hyperparameter. Analogously, a preference is more likely to be missing in the training data if the item is less popular. To compensate for this bias, Steck proposes to weight the missing preferences non-uniformly:

$$\mathbf{W}_{ui} = \mathbf{R}_{ui} 1 + (1 - \mathbf{R}_{ui}) \cdot C \cdot c(i)^\beta, \tag{2.25}$$

with C a constant. Steck proposed these two weighting strategies as alternatives to each other. However, we believe that they can be combined since they are the application of the same idea to the known and missing preferences respectively.

Although they provide less motivation, Pan et al. [65] arrive at similar weighting schemes. They propose $\mathbf{W}_{ui} = 1$ if $\mathbf{R}_{ui} = 1$ and give three possibilities for the case when $\mathbf{R}_{ui} = 0$:

$$\mathbf{W}_{ui} = \delta, \tag{2.26}$$

$$\mathbf{W}_{ui} = \alpha \cdot c(u), \tag{2.27}$$

$$\mathbf{W}_{ui} = \alpha \left(|\mathcal{U}| - c(i)\right), \tag{2.28}$$

with $\delta \in [0,1]$ a uniform hyperparameter and α a hyperparameter such that $\mathbf{W}_{ui} \leq 1$ for all pairs (u,i) for which $\mathbf{R}_{ui} = 0$. In the first case, all missing preferences get the same weight. In the second case, a missing preference is more negative if the user already has many preferences. In the third case, a missing preference is less negative if the item is popular. Interestingly, the third weighting scheme is orthogonal to the one of Steck in Equation 2.25. Additionally, Pan et al. [65] propose a deviation function with an alternative regularization:

$$\mathcal{D}(\theta, \mathbf{R}) = \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} \mathbf{W}_{ui} \left((\mathbf{R}_{ui} - \mathbf{S}_{ui})^2 + \lambda \left(||\theta_u||_F^2 + ||\theta_i||_F^2 \right) \right). \tag{2.29}$$

Yao et al. [114] adopt a more complex missing data model that has a hyperparameter p, which indicates the overall likelihood that a missing preference is preferred. This translates into the deviation function:

$$\begin{aligned} &\sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} \mathbf{R}_{ui} \mathbf{W}_{ui} (1 - \mathbf{S}_{ui})^2 \\ &+ \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} (1 - \mathbf{R}_{ui}) \mathbf{W}_{ui} (p - \mathbf{S}_{ui})^2 \\ &+ \sum_{t=1}^{T} \sum_{f=1}^{F} \lambda_{tf} ||\mathbf{S}^{(t,f)}||_F^2 \end{aligned} \tag{2.30}$$

The special case with $p = 0$ reduces this deviation function to the one in Equation 2.24.

An even more complex missing data model and corresponding deviation function was proposed by Sindhwani et al. [85]:

$$\begin{aligned} &\sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} \mathbf{R}_{ui} \mathbf{W}_{ui} (1 - \mathbf{S}_{ui})^2 \\ &+ \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} (1 - \mathbf{R}_{ui}) \mathbf{W}_{ui} (\mathbf{P}_{ui} - \mathbf{S}_{ui})^2 \\ &+ \sum_{t=1}^{T} \sum_{f=1}^{F} \lambda_{tf} ||\mathbf{S}^{(t,f)}||_F^2 \\ &- \lambda_H \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} (1 - \mathbf{R}_{ui}) H(\mathbf{P}_{ui}) \end{aligned} \tag{2.31}$$

with $\mathbf{P}$ and $\mathbf{W}$ additional parameters of the model that need to be computed based on the data $\mathbf{R}$, together with all other parameters. The last term contains the entropy

function H and serves as a regularizer for $\mathbf{P}$. Furthermore, they define the constraint

$$\frac{1}{|\mathcal{U}||\mathcal{I}| - |\mathcal{R}|} \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} \mathbf{P}_{ui} = p,$$

which expresses that the average probability that a missing value is actually one must be equal to the hyperparameter p. To reduce the computational cost, they fix $\mathbf{P}_{ui} = 0$ for most (u,i)-pairs and randomly choose a few (u,i)-pairs for which $\mathbf{P}_{ui}$ is computed based on the data. It seems that this simplification completely offsets the modeling flexibility that was obtained by introducing $\mathbf{P}$. Additionally, they simplify $\mathbf{W}$ as the one-dimensional matrix factorization

$$\mathbf{W}_{ui} = \mathbf{V}_u \mathbf{V}_i.$$

A conceptual inconsistency of this deviation function is that although the recommendation score is given by $\mathbf{S}_{ui}$, also $\mathbf{P}_{ui}$ could be used. Hence, there exist two parameters for the same concept, which is ambiguous at least.

2.3.4 Maximum Margin-based

A disadvantage of the squared error-based deviation functions is their symmetry. For example, if $\mathbf{R}_{ui} = 1$ and $\mathbf{S}_{ui} = 0$, $(\mathbf{R}_{ui} - \mathbf{S}_{ui})^2 = 1$. This is desirable behavior because we want to penalize the model for predicting that a preference is not preferred. However if $\mathbf{S}_{ui} = 2$, we obtain the same penalty: $(\mathbf{R}_{ui} - \mathbf{S}_{ui})^2 = 1$. This, on the other hand, is not desirable because we do not want to penalize the model for predicting that a preference will definitely be preferred.

A maximum margin-based deviation function does not suffer from this problem [64]:

$$\mathcal{D}(\theta, \mathbf{R}) = \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} \mathbf{W}_{ui} \cdot h\left(\tilde{\mathbf{R}}_{ui} \cdot \mathbf{S}_{ui}\right) + \lambda ||\mathbf{S}||_{\Sigma}, \tag{2.32}$$

with $||.||_{\Sigma}$ the trace norm, λ a regularization hyperparameter, $h\left(\tilde{\mathbf{R}}_{ui} \cdot \mathbf{S}_{ui}\right)$ a smooth hinge loss given by Figure 2.2 [76], $\mathbf{W}$ given by one of the Equations 2.26-2.28 and the matrix $\tilde{\mathbf{R}}$ defined as

$$\begin{cases} \tilde{\mathbf{R}}_{ui} = 1 & if\, \mathbf{R}_{ui} = 1 \\ \tilde{\mathbf{R}}_{ui} = -1 & if\, \mathbf{R}_{ui} = 0. \end{cases}$$

This deviation function incorporates the confidence about the training data by means of $\mathbf{W}$ and the missing knowledge about the degree of preference by means of the hinge loss $h\left(\tilde{\mathbf{R}}_{ui} \cdot \mathbf{S}_{ui}\right)$. Since the degree of a preference $\mathbf{R}_{ui} = 1$ is considered unknown, a value $\mathbf{S}_{ui} > 1$ is not penalized if $\mathbf{R}_{ui} = 1$.

2.3.5 Overall Ranking Error-based

The scores $\mathbf{S}$ computed by a collaborative filtering method are used to personally rank all items for every user. Therefore, one can argue that it is more natural to directly optimize the ranking of the items instead of their individual scores.

Rendle et al. [75], aim to maximize the area under the ROC curve (AUC), which is given by:

$$AUC = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{|u| \cdot (|\mathcal{I}| - |u|)} \sum_{\mathbf{R}_{ui}=1} \sum_{\mathbf{R}_{uj}=0} \mathbb{I}(\mathbf{S}_{ui} > \mathbf{S}_{uj}). \tag{2.33}$$

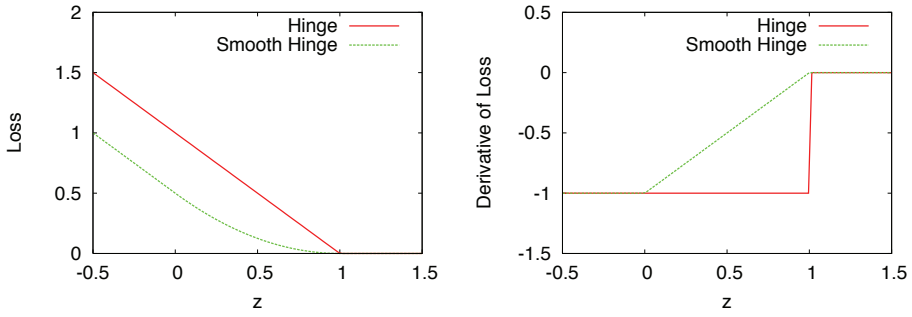

Figure 2.2: Shown are the loss function values $h(z)$ (left) and the gradients $dh(z)/dz$ (right) for the Hinge and Smooth Hinge. Note that the gradients are identical outside the region $z \in (0,1)$) [76].

If the AUC is higher, the pairwise rankings induced by the model $\mathbf{S}$ are more in line with the observed data $\mathbf{R}$. However, because $\mathbb{I}(\mathbf{S}_{ui} > \mathbf{S}_{uj})$ is non-differentiable, it is impossible to actually compute the parameters that (locally) maximize the AUC. Their solution is a deviation function, called the *Bayesian Personalized Ranking(BPR)-criterium*, which is a differentiable approximation of the negative AUC from which constant factors have been removed and to which a regularization term has been added:

$$\mathcal{D}(\theta, \mathbf{R}) = \sum_{u \in \mathcal{U}} \sum_{\mathbf{R}_{ui}=1} \sum_{\mathbf{R}_{uj}=0} -\log \sigma(\mathbf{S}_{ui} - \mathbf{S}_{uj}) + \sum_{t=1}^{T} \sum_{f=1}^{F} \lambda_{tf} ||\mathbf{S}^{(t,f)}||_F^2, \tag{2.34}$$

with $\sigma(\cdot)$ the sigmoid function and λ_{tf} regularization hyperparameters. Since this approach explicitly accounts for the missing data, it corresponds to the MNAR assumption.

Pan and Chen claim that it is beneficial to relax the BPR deviation function by Rendle et al. to account for noise in the data [66, 67]. Specifically, they propose CoFiSet [66], which allows certain violations $\mathbf{S}_{ui} < \mathbf{S}_{uj}$ when $\mathbf{R}_{ui} > \mathbf{R}_{uj}$:

$$\mathcal{D}(\theta, \mathbf{R}) = \sum_{u \in \mathcal{U}} \sum_{I \subseteq I(u)} \sum_{J \subseteq \mathcal{I} \setminus I(u)} -\log \sigma \left(\frac{\sum_{i \in I} \mathbf{S}_{ui}}{|I|} - \frac{\sum_{j \in J} \mathbf{S}_{uj}}{|J|} \right) + \Gamma(\theta), \tag{2.35}$$

with $\Gamma(\theta)$ a regularization term that slightly deviates from the one proposed by Rendle et al. for no clear reason. Alternatively, they propose GBPR [67], which relaxes the BPR deviation function in a different way:

$$\mathcal{D}(\theta, \mathbf{R}) = \sum_{u \in \mathcal{U}} \sum_{\mathbf{R}_{ui}=1} \sum_{\mathbf{R}_{uj}=0} -\log \sigma^2 \left(\alpha \cdot \frac{\sum_{g \in \mathcal{G}_{u,i}} \mathbf{S}_{gi}}{|\mathcal{G}_{u,i}|} + (1-\alpha) \cdot \mathbf{S}_{ui} - \mathbf{S}_{uj} \right) + \Gamma(\theta), \tag{2.36}$$

with $\mathcal{G}_{u,i}$ the union of $\{u\}$ with a random subset of $\{g \in \mathcal{U} \setminus \{u\} | \mathbf{R}_{gi} = 1\}$, and α a hyperparameter.

Furthermore, also Kabbur et al. [46] aim to maximize AUC with their method FISMauc. However, they propose to use a different differentiable approximation of

AUC:

$$\mathcal{D}(\theta,\mathbf{R}) = \sum_{u\in\mathcal{U}}\sum_{\mathbf{R}_{ui}=1}\sum_{\mathbf{R}_{uj}=0}\left((\mathbf{R}_{ui}-\mathbf{R}_{uj})-(\mathbf{S}_{ui}-\mathbf{S}_{uj})\right)^2 + \sum_{t=1}^{T}\sum_{f=1}^{F}\lambda_{tf}||\mathbf{S}^{(t,f)}||_F^2. \quad (2.37)$$

The same model without regularization was proposed by Töscher et al. [96]:

$$\mathcal{D}(\theta,\mathbf{R}) = \sum_{u\in\mathcal{U}}\sum_{\mathbf{R}_{ui}=1}\sum_{\mathbf{R}_{uj}=0}\left((\mathbf{R}_{ui}-\mathbf{R}_{uj})-(\mathbf{S}_{ui}-\mathbf{S}_{uj})\right)^2. \quad (2.38)$$

A similar deviation function was proposed by Takács and Tikk [93]:

$$\mathcal{D}(\theta,\mathbf{R}) = \sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}}\mathbf{R}_{ui}\sum_{j\in\mathcal{I}}w(j)\left((\mathbf{S}_{ui}-\mathbf{S}_{uj})-(\mathbf{R}_{ui}-\mathbf{R}_{uj})\right)^2, \quad (2.39)$$

with $w(\cdot)$ a user-defined item weighting function. The simplest choice is $w(j)=1$ for all j. An alternative proposed by Takács and Tikk is $w(j)=c(j)$. Another important difference is that this deviation function also minimizes the score-difference between the known preferences mutually.

Finally, it is remarkable that both Töscher, and Takács and Tikk explicitly do not add a regularization term, whereas most other authors find that the regularization term is important for their model's performance.

2.3.6 Top of Ranking Error-based

Very often, only the N highest ranked items are shown to users. Therefore, Shi et al. [83] propose to improve the top of the ranking at the cost of worsening the overall ranking. Specifically, they propose the CLiMF method, which aims to minimize the *mean reciprocal rank (MRR)* instead of the AUC. The MRR is defined as

$$MRR = \frac{1}{|\mathcal{U}|}\sum_{u\in\mathcal{U}} r_>\left(\max_{\mathbf{R}_{ui}=1}\mathbf{S}_{ui} \mid \mathbf{S}_{u\cdot}\right)^{-1},$$

in which $r_>(a|\mathbf{b})$ gives the rank of a among all numbers in $\mathbf{b}$ when ordered in descending order. Unfortunately, the non-smoothness of $r_>()$ and max makes the direct optimization of MRR unfeasible. Hence, Shi et al. derive a differentiable version of MRR. Yet, optimizing it is intractable. Therefore, they propose to optimize a lower bound instead. After also adding regularization terms, their final deviation function is given by

$$\begin{aligned}\mathcal{D}(\theta,\mathbf{R}) = -\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}}\mathbf{R}_{ui}\Big(&\log\sigma(\mathbf{S}_{ui}) \\ &+\sum_{j\in\mathcal{I}}\log\left(1-\mathbf{R}_{uj}\sigma(\mathbf{S}_{uj}-\mathbf{S}_{ui})\right)\Big) \\ &+\sum_{t=1}^{T}\sum_{f=1}^{F}\lambda_{tf}||\mathbf{S}^{(t,f)}||_F^2,\end{aligned} \quad (2.40)$$

with λ a regularization constant and $\sigma(\cdot)$ the sigmoid function. A disadvantage of this deviation function is that it ignores all missing data, i.e. it corresponds to the MAR assumption.

An alternative for MRR is *mean average precision (MAP)*:

$$MAP = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{c(u)} \sum_{\mathbf{R})_{ui}=1} \frac{1}{r_{>}(\mathbf{S}_{ui} | \mathbf{S}_{u\cdot})} \sum_{\mathbf{R}_{uj}=1} \mathbb{I}(\mathbf{S}_{uj} \geq \mathbf{S}_{ui}),$$

which still emphasizes the top of the ranking, but less extremely than MRR. However, also MAP is non-smooth, preventing its direct optimization. Also for MAP, Shi et al. derived a differentiable version, called TFMAP [82]:

$$\mathcal{D}(\theta, \mathbf{R}) = -\sum_{u \in \mathcal{U}} \frac{1}{c(u)} \sum_{\mathbf{R})_{ui}=1} \sigma(\mathbf{S}_{ui}) \sum_{\mathbf{R}_{uj}=1} \sigma\left(\mathbf{S}_{uj} - \mathbf{S}_{ui}\right) + \lambda \cdot \sum_{t=1}^{T} \sum_{f=1}^{F} ||\mathbf{S}^{(t,f)}||_F^2, \quad (2.41)$$

with λ a regularization hyperparameter. Besides, the formulation of TFMAP in Equation 2.41 is a simplified version of the original, concieved for multi-context data, which is out of the scope of this thesis.

Dhanjal et al. proposed yet another alternative [21]. They start from the definition of *AUC* in Equation 2.33, approximate the indicator function $\mathbb{I}(\mathbf{S}_{ui} - \mathbf{S}_{uj})$ by the squared hinge loss $\left(\max\left(0, 1 + \mathbf{S}_{uj} - \mathbf{S}_{ui}\right)\right)^2$ and emphasize the deviation at the top of the ranking by means of the hyperbolic tangent function $\tanh(\cdot)$:

$$\mathcal{D}(\theta, \mathbf{R}) = \sum_{u \in \mathcal{U}} \sum_{\mathbf{R}_{ui}=1} \tanh\left(\sum_{\mathbf{R}_{uj}=0} \left(\max\left(0, 1 + \mathbf{S}_{uj} - \mathbf{S}_{ui}\right)\right)^2 \right). \quad (2.42)$$

2.3.7 k-Order Statistic-based

On the one hand, the deviation functions in Equations 2.34-2.39 try to minimize the overall rank of the known preferences. On the other hand, the deviation functions in Equations 2.40 and 2.42 try to push one or a few known preferences as high as possible to the top of the item-ranking. Weston et al. [110] propose to minimize a trade-off between the above two extremes:

$$\sum_{u \in \mathcal{U}} \sum_{\mathbf{R}_{ui}=1} w\left(\frac{r_{>}(\mathbf{S}_{ui} \mid \{\mathbf{S}_{ui} \mid \mathbf{R}_{ui} = 1\})}{c(u)} \right) \sum_{\mathbf{R}_{uj}=0} \frac{\mathbb{I}(\mathbf{S}_{uj} + 1 \geq \mathbf{S}_{ui})}{r_{>}(\mathbf{S}_{uj} \mid \{\mathbf{S}_{uk} \mid \mathbf{R}_{uk} = 0\})}, \quad (2.43)$$

with $w(\cdot)$ a function that weights the importance of the known preference as a function of their predicted rank among all known preferences. This weighting function is user-defined and determines the trade-off between the two extremes, i.e. minimizing the mean rank of the known preferences and minimizing the maximal rank of the known preferences. Because this deviation function is non-differentiable, Weston et al. propose the differentiable approximation

$$\mathcal{D}(\theta, \mathbf{R}) = \sum_{u \in \mathcal{U}} \sum_{\mathbf{R}_{ui}=1} w\left(\frac{r_{>}(\mathbf{S}_{ui} \mid \{\mathbf{S}_{ui} \mid \mathbf{R}_{ui} = 1\})}{c(u)} \right) \sum_{\mathbf{R}_{uj}=0} \frac{\max(0, 1 + \mathbf{S}_{uj} - \mathbf{S}_{ui})}{N^{-1}\left(|\mathcal{I}| - c(u)\right)}, \quad (2.44)$$

in which they replaced the indicator function by the hinge-loss and approximated the rank with $N^{-1}\left(|\mathcal{I}| - c(u)\right)$, in which N the number of items k that were randomly sampled until $\mathbf{S}_{uk} + 1 > \mathbf{S}_{ui}$[4]. Furthermore, Weston et al. use the simple weighting

[4] Weston et al. [108] provide a justification for this approximation.

function

$$\begin{cases} w\left(\frac{r_>(\mathbf{S}_{ui}|\{\mathbf{S}_{ui}|\mathbf{R}_{ui}=1\})}{|u|}\right) = 1 \textbf{ if } r_>(\mathbf{S}_{ui} \mid S \subseteq \{\mathbf{S}_{ui} \mid \mathbf{R}_{ui}=1\}, |S|=K) = k \textbf{ and} \\ w\left(\frac{r_>(\mathbf{S}_{ui}|\{\mathbf{S}_{ui}|\mathbf{R}_{ui}=1\})}{|u|}\right) = 0 \textbf{ otherwise}, \end{cases}$$

i.e. from the set S that contains K randomly sampled known preferences, ranked by their predicted score, only the item at rank k is selected to contribute to the training error. When k is set low, the top of the ranking will be optimized at the cost of a worse mean rank. When k is set higher, the mean rank will be optimized at the cost of a worse top of the ranking. The regularization is not done by adding a regularization term but by forcing the norm of the factor matrices to be below a maximum, which is a hyperparameter.

Alternatively, Weston et al. also propose a simplified deviation function:

$$\mathcal{D}(\theta, \mathbf{R}) = \sum_{u \in \mathcal{U}} \sum_{\mathbf{R}_{ui}=1} w\left(\frac{r_>(\mathbf{S}_{ui} \mid \{\mathbf{S}_{ui} \mid \mathbf{R}_{ui}=1\})}{c(u)}\right) \sum_{\mathbf{R}_{uj}=0} \max(0, 1 + \mathbf{S}_{uj} - \mathbf{S}_{ui}). \quad (2.45)$$

2.3.8 Rank Link Function-based

The ranking-based deviation functions discussed so far, are all tailor made differentiable approximations with respect to the recommendation scores of a certain ranking quality measure, like AUC, MRR or the k-th order statistic. Steck [89] proposes a more general approach that is applicable to any ranking quality measure that is differentiable with respect to the rankings. He demonstrates his method on two ranking quality measures: *AUC* and *nDCG*. For AUC_u, the *AUC* for user u, he does not use the formulation in Equation 2.33, but uses the equivalent

$$AUC_u = \frac{1}{c(u) \cdot (|\mathcal{I}| - c(u))} \left[(|\mathcal{I}|+1)c(u) - \binom{c(u)+1}{2} - \sum_{\mathbf{R}_{ui}=1} r_{ui} \right],$$

with r_{ui} the rank of item i in the recommendation list of user u. Second, $nDCG_u$ is defined as

$$nDCG_u = \frac{DCG}{DCG_{opt}},$$
$$DCG = \sum_{\mathbf{R}_{ui}=1} \frac{1}{\log(r_{ui}+1)},$$

with DCG_{opt} the *DCG* of the optimal ranking. In both cases, Steck proposes to relate the rank r_{ui} with the recommendation score $\mathbf{R}_{ui}$ by means of a link function

$$r_{ui} = \max\left\{1, |\mathcal{I}| \cdot \left(1 - cdf(\hat{\mathbf{S}}_{ui})\right)\right\}, \quad (2.46)$$

with $\hat{\mathbf{S}}_{ui} = (\mathbf{S}_{ui} - \mu_u)/std_u$ the normalized recommendation score in which μ_u and std_u are the mean and standard deviation of the recommendation scores for user u, and *cdf* is the cumulative distribution of the normalized scores. However, to know *cdf*, he needs to assume a distribution for the normalized recommendation scores. He motivates that a zero-mean normal distribution of the recommendation scores is a reasonable assumption. Consequently, $cdf(\hat{\mathbf{S}}_{ui}) = probit(\hat{\mathbf{S}}_{ui})$. Furthermore,

he proposes to approximate the probit function with the computationally more efficient sigmoid function or the even more efficient function

$$rankSE(\hat{\mathbf{S}}_{ui}) = [1 - ([1 - \hat{\mathbf{S}}_{ui}]_+)^2]_+,$$

with $[x]_+ = max\{0, x\}$. In his *pointwise* approach, Steck uses Equation 2.46 to compute the ranks based on the recommendation scores. In his *listwise* approach, on the other hand, he finds the actual rank of a recommendation score by sorting the recommendation scores for every user, and uses Equation 2.46 only to compute the gradient of the rank with respect to the recommendation score during the minimization procedure. After adding regularizaton terms, he proposes the deviation function

$$\mathcal{D}(\theta, \mathbf{R}) = \sum_{u \in \mathcal{U}} \left(\sum_{\mathbf{R}_{ui}=1} \left(L(\mathbf{S}_{ui}) + \lambda \cdot \left(||\theta_u||_F^2 + ||\theta_i||_F^2 \right) \right) + \sum_{j \in \mathcal{I}} \gamma \cdot [\mathbf{S}_{uj}]_+^2 \right), \tag{2.47}$$

with θ_u, θ_i the vectors that group all model parameters related to user u and item i respectively, λ, γ regularization hyperparameters, and $L(\mathbf{S}_{ui})$ equal to $-AUC_u$ or $-nDCG_u$, which are a function of $\mathbf{S}_{ui}$ via r_{ui}. The last regularization term is introduced to enforce an approximately normal distribution on the scores.

2.3.9 Posterior KL-divergence-based

In our framework, the *optimal* parameters θ^* are computed as

$$\theta^* = \arg\min_{\theta} \mathcal{D}(\theta, \mathbf{R}).$$

However, we can consider this a special case of

$$\theta^* = \psi \left(\arg\min_{\phi} \mathcal{D}\left(\theta(\phi), \mathbf{R}\right) \right),$$

in which ψ is chosen to be the identity function and the parameters θ are identical to the parameters ϕ, i.e. $\theta(\phi) = \phi$. Now, some authors [47, 68, 29] propose to choose $\psi()$ and ϕ differently.

Specifically, they model every parameter θ_j of the factorization model as a random variable with a parameterized posterior probability density function $p(\theta_j|\phi_j)$. Hence, finding the variables ϕ corresponds to finding the posterior distributions of the model parameters θ.

Because it is intractable to find the true posterior distribution $p(\theta|\mathbf{R})$ of the parameters, they settle for a *mean-field* approximation $q(\theta|\phi)$, in which all variables are assumed to be independent.

Then, they define ψ as $\psi(\phi^*) = \mathbb{E}_{q(\theta|\phi^*)}[\theta]$, i.e. the point-estimate of the parameters θ equals their expected value under the mean-field approximation of their posterior distributions. Notice that $\theta_j^* = \mathbb{E}_{q(\theta_j|\phi_j^*)}[\theta_j]$ because of the independence assumption.

If all parameters θ_j are assumed to be normally distributed with mean μ_j and variance σ_j^2 [47, 68], $\phi_j = (\mu_j, \sigma_j)$ and $\theta_j^* = \mathbb{E}_{q(\theta_j|\phi_j^*)}[\theta_j] = \mu_j^*$. If, on the other hand, all parameters θ_j are assumed to be gamma distributed with shape α_j and rate β_j [29], $\phi_j = (\alpha_j, \beta_j)$ and $\theta_j^* = \mathbb{E}_{q(\theta_j|\phi_j^*)}[\theta_j] = \alpha_j^*/\beta_j^*$.

Furthermore, prior distributions are defined for all parameters θ. Typically, when this approach is adopted, the underlying assumptions are represented as a *graphical model* [11].

The parameters ϕ, and therefore the corresponding mean-field approximations of the posterior distribution of the parameters θ, can be inferred by defining the deviation function as the KL-divergence of the real (unknown) posterior distribution of the parameters, $p(\theta|\mathbf{R})$, from the modeled posterior distribution of the parameters, $q(\theta|\phi)$,:

$$\mathcal{D}\left(\theta(\phi), \mathbf{R}\right) = D_{KL}\left(q(\theta|\phi) \| p(\theta|\mathbf{R})\right),$$

which can be solved despite the fact that $p(\theta|\mathbf{R})$ is unknown [44]. This approach goes by the name *variational inference* [44].

A *nonparametric* version of this approach also considers D, the number of latent dimensions in the simple two factor factorization model of Equation 2.6, as a parameter that depends on the data $\mathbf{R}$ instead of a hyperparameter, as most other methods do [30].

Notice that certain solutions for latent Dirichlet allocation [12] also use variational inference techniques. However, in this case, variational inference is a part of the (variational) expectation-maximization algorithm for computing the parameters that optimize the negative log-likelihood of the model parameters, which serves as the deviation function. This is different from the methods discussed in this section, where the KL-divergence between the real and the approximate posterior is the one and only deviation function.

2.3.10 Convex

An intuitively appealing but non-convex deviation function is worthless if there exists no algorithm that can efficiently compute parameters that correspond to a good local minimum of this deviation function. Convex deviation functions on the other hand, have only one global minimum that can be computed with one of the well studied convex optimization algorithms. For this reason, it is worthwhile to pursue convex deviation functions.

Aiolli [2] proposes a convex deviation function based on the AUC (Eq. 2.33). For every individual user $u \in \mathcal{U}$, Aiolli starts from AUC_u, the AUC for u:

$$AUC_u = \frac{1}{c(u) \cdot (|\mathcal{I}| - c(u))} \sum_{\mathbf{R}_{ui}=1} \sum_{\mathbf{R}_{uj}=0} \mathbb{I}(\mathbf{S}_{ui} > \mathbf{S}_{uj}).$$

Next, he proposes a lower bound on AUC_u:

$$AUC_u \geq \frac{1}{c(u) \cdot (|\mathcal{I}| - c(u))} \sum_{\mathbf{R}_{ui}=1} \sum_{\mathbf{R}_{uj}=0} \frac{\mathbf{S}_{ui} - \mathbf{S}_{uj}}{2},$$

and interprets it as a weighted sum of margins $\frac{\mathbf{S}_{ui}-\mathbf{S}_{uj}}{2}$ between any known preferences and any absent feedback, in which every margin gets the same weight $\frac{1}{c(u)\cdot(|\mathcal{I}|-c(u))}$. Hence maximizing this lower bound on the AUC corresponds to maximizing the sum of margins between any known preference and any absent feedback in which every margin has the same weight. A problem with maximizing this sum is

that very high margins on pairs that are easily ranked correctly, can hide poor (negative) margins on pairs that are difficult to rank correctly. Therefore, Aiolli proposes to replace the uniform weighting scheme with a weighting scheme that emphasizes the difficult pairs such that the total margin is the worst possible case, i.e. the lowest possible sum of weighted margins. Specifically, he propose to solve for every user u, the joint optimization problem

$$\theta^* = \arg\max_{\theta} \min_{\alpha_{u*}} \sum_{\mathbf{R}_{ui}=1} \sum_{\mathbf{R}_{uj}=0} \alpha_{ui}\alpha_{uj}(\mathbf{S}_{ui} - \mathbf{S}_{uj}),$$

where for every user u, it holds that $\sum_{\mathbf{R}_{ui}=1} \alpha_{ui} = 1$ and $\sum_{\mathbf{R}_{uj}=0} \alpha_{uj} = 1$. To avoid overfitting of α, he adds two regularization terms:

$$\theta^* = \arg\max_{\theta} \min_{\alpha_{u*}} \left(\sum_{\mathbf{R}_{ui}=1} \sum_{\mathbf{R}_{uj}=0} \alpha_{ui}\alpha_{uj}(\mathbf{S}_{ui} - \mathbf{S}_{uj}) + \lambda_p \sum_{\mathbf{R}_{ui}=1} \alpha_{ui}^2 + \lambda_n \sum_{\mathbf{R}_{ui}=0} \alpha_{ui}^2 \right),$$

with λ_p, λ_n regularization hyperparameters. The model parameters θ, on the other hand, are regularized by normalization constraints on the factor matrices. Solving the above maximization for every user, is equivalent to minimizing the deviation function

$$\mathcal{D}(\theta, \mathbf{R}) = \sum_{u \in \mathcal{U}} \left(\max_{\alpha_{u*}} \left(\sum_{\mathbf{R}_{ui}=1} \sum_{\mathbf{R}_{uj}=0} \alpha_{ui}\alpha_{uj}(\mathbf{S}_{uj} - \mathbf{S}_{ui}) - \lambda_p \sum_{\mathbf{R}_{ui}=1} \alpha_{ui}^2 - \lambda_n \sum_{\mathbf{R}_{ui}=0} \alpha_{ui}^2 \right) \right). \tag{2.48}$$

2.3.11 Analytically Solvable

Some deviation functions are not only convex, but also analytically solvable. This means that the parameters that minimize these deviation functions can be exactly computed from a formula and that no numerical optimization algorithm is required.

Traditionally, the methods that adopt these deviation functions have been inappropriately called *neighborhood* or *memory-based* methods. First, although these methods adopt neighborhood-based factorization models, there are also neighborhood-based methods that adopt non-convex deviation functions. Examples are SLIM [60] and BPR-kNN [75], which were, amongst others, discussed in Section 2.2. Second, a *memory-based* implementation of these methods, in which the necessary parameters of the factorization model are not precomputed, but computed in real time when they are required is conceptually possible, yet practically intractable in most cases. Instead, a *model-based* implementation of these methods, in which the factorization model is precomputed, is the best choice for the majority of applications.

Basic Neighborhood-based

A first set of analytically solvable deviation functions is tailored to the item-based neighborhood factorization models of Equation 2.9:

$$\mathbf{S} = \mathbf{R}\mathbf{S}^{(1,2)}.$$

As explained in Section 2.2, the factor matrix $\mathbf{S}^{(1,2)}$ can be interpreted as an item-similarity matrix. Consequently, these deviation functions compute every parameter in $\mathbf{S}^{(1,2)}$ as

$$\mathbf{S}^{(1,2)}_{ji} = sim(j,i),$$

with $sim(j,i)$ the similarity between items j and i according to some analytically computable similarity function. This is equivalent to

$$\mathbf{S}^{(1,2)}_{ji} - sim(j,i) = 0,$$

which is true for all (j,i)-pairs if and only if

$$\sum_{j\in\mathcal{I}}\sum_{i\in\mathcal{I}}\left(\mathbf{S}^{(1,2)}_{ji} - sim(j,i)\right)^2 = 0.$$

Hence, computing the factor matrix $\mathbf{S}^{(1,2)}_{ji}$ corresponds to minimizing the deviation function

$$\mathcal{D}(\theta,\mathbf{R}) = \sum_{j\in\mathcal{I}}\sum_{i\in\mathcal{I}}\left(\mathbf{S}^{(1,2)}_{ji} - sim(j,i)\right)^2.$$

In this case, the deviation function mathematically expresses the interpretation that $sim(j,i)$ is a good predictor for preferring i if j is also preferred. The key property that determines th analytical solvability of this deviation function is the absence of products of parameters. The non-convex deviation functions in Section 2.4.1, on the other hand, do contain products of parameters, which are contained in the term $\mathbf{S}_{ui}$. Consequently, they are harder to solve but allow richer parameter interactions.

A typical choice for $sim(j,i)$ is the cosine similarity [19]. The cosine similarity between two items j and i is given by:

$$cos(j,i) = \frac{\sum_{v\in\mathcal{U}}\mathbf{R}_{vj}\mathbf{R}_{vi}}{\sqrt{c(i)\cdot c(j)}}. \tag{2.49}$$

Another similarity measure is the conditional probability similarity measure [19], which is for two items i and j given by:

$$condProb(j,i) = \sum_{v\in\mathcal{U}}\frac{\mathbf{R}_{vi}\mathbf{R}_{vj}}{c(j)}. \tag{2.50}$$

Deshpande et al. [19] also proposed an adapted version:

$$condProb^*(j,i) = \sum_{v\in\mathcal{U}}\frac{\mathbf{R}_{vi}\mathbf{R}_{vj}}{c(i)\cdot c(j)^\alpha\cdot c(v)}, \tag{2.51}$$

in which $\alpha\in[0,1]$ a hyperparameter. They introduced the factor $1/c(j)^\alpha$ to avoid the recommendation of overly frequent items and the factor $1/c(v)$ to reduce the weight of i and j co-occurring in the preferences of v, if v has more preferences. Other similarity measures were proposed by Aiolli [1]:

$$sim(j,i) = \left(\sum_{v\in\mathcal{U}}\frac{\mathbf{R}_{vi}\mathbf{R}_{vj}}{c(j)^\alpha\cdot c(i)^{(1-\alpha)}}\right)^q,$$

with α, q hyperparameters, Gori and Pucci [31]:

$$sim(j,i) = \frac{\sum_{v \in \mathcal{U}} \mathbf{R}_{vj}\mathbf{R}_{vi}}{\sum_{k \in \mathcal{I}} \sum_{v \in \mathcal{U}} \mathbf{R}_{vj}\mathbf{R}_{vk}},$$

and Wang et al. [106]:

$$sim(j,i) = \log\left(1 + \alpha \cdot \frac{\sum_{v \in \mathcal{U}} \mathbf{R}_{vj}\mathbf{R}_{vi}}{c(j)c(i)}\right),$$

with $\alpha \in \mathbb{R}_0^+$ a hyperparameter. Furthermore, Huang et al. show that $sim(j,i)$ can also be chosen from a number of similarity measures that are typically associated with link prediction [39]. Similarly, Bellogin et al. show that also typical scoring functions used in information retrieval can be used for $sim(j,i)$ [8].

It is common practice to introduce sparsity in $\mathbf{S}^{(1,2)}$ by defining

$$sim(j,i) = sim'(j,i) \cdot |top\text{-}k(j) \cap \{i\}|, \tag{2.52}$$

with $sim'(j,i)$ one of the similarity functions defined by Equations 2.49-2.51, $top\text{-}k(j)$ the set of items l that correspond to the k highest values $sim'(j,l)$, and k a hyperparameter.

Motivated by a qualitative examination of their results, Sigurbjörnsson and Van Zwol [84] proposed additional adaptations:

$$sim(j,i) = s(j) \cdot d(i) \cdot r(j,i) \cdot sim'(j,i) \cdot |top\text{-}k(j) \cap \{i\}|,$$

with

$$s(j) = \frac{k_s}{k_s + |k_s - \log c(j)|}, \tag{2.53}$$

$$d(i) = \frac{k_d}{k_d + |k_d - \log c(i)|}, \tag{2.54}$$

$$r(j,i) = \frac{k_r}{k_r + (r-1)}, \tag{2.55}$$

in which i is the r-th most similar item to j and k_s, k_d and k_r are hyperparameters.

Finally, Desphande and Karypis [19] proposes to normalize $sim(j,i)$ as

$$sim(j,i) = \frac{sim''(j,i)}{\sum_{l \in \mathcal{I} \setminus \{j\}} sim''(j,l)},$$

with $sim''(j,i)$ defined by Equation 2.52. Alternatively, Aiolli [1] proposes the normalization

$$sim(j,i) = \frac{sim''(j,i)}{\sum_{l \in \mathcal{I} \setminus \{i\}} sim''(l,i)^{2(1-\beta)}},$$

with β a hyperparameter.

A second set of analytically solvable deviation functions is tailored to the user-based neighborhood factorization model of Equation 2.12:

$$\mathbf{S} = \mathbf{S}^{(1,1)}\mathbf{R}.$$

In this case, the factor matrix $\mathbf{S}^{(1,1)}$ can be interpreted as a user-similarity matrix. Consequently, these deviation functions compute every parameter in $\mathbf{S}^{(1,1)}$ as

$$\mathbf{S}^{(1,1)}_{uv} = sim(u,v),$$

with $sim(u,v)$ the similarity between users u and v according to some analytically computable similarity function. In the same way as for the item-based case, computing the factor matrix $\mathbf{S}^{(1,1)}_{uv}$ corresponds to minimizing the deviation function

$$\mathcal{D}(\theta,\mathbf{R}) = \sum_{u\in\mathcal{U}}\sum_{v\in\mathcal{U}}\left(\mathbf{S}^{(1,1)}_{uv} - sim(u,v)\right)^2.$$

In this case, the deviation function mathematically expresses the interpretation that users u and v for which $sim(u,v)$ is high, prefer the same items.

Sarwar et al. [79] propose

$$sim(u,v) = |top\text{-}k(u) \cap \{v\}|,$$

with $top\text{-}k(u)$ the set of users w that have the k highest cosine similarities $\cos(u,w)$ with user u, and k a hyperparameter. In this case, cosine similarity is defined as

$$cos(u,v) = \frac{\sum_{j\in\mathcal{I}}\mathbf{R}_{uj}\mathbf{R}_{vj}}{\sqrt{c(u)\cdot c(v)}}. \tag{2.56}$$

Alternatively, Aiolli [1] proposes

$$sim(u,v) = \left(\sum_{j\in\mathcal{I}}\frac{\mathbf{R}_{uj}\mathbf{R}_{vj}}{c(u)^{\alpha}\cdot c(v)^{(1-\alpha)}}\right)^q,$$

with α, q hyperparameters, and Wang et al. [106] propose

$$sim(u,v) = \log\left(1+\alpha\cdot\frac{\sum_{j\in\mathcal{U}}\mathbf{R}_{uj}\mathbf{R}_{vj}}{c(u)c(v)}\right),$$

with α a hyperparameter.

The deviation function function for the unified neighborhood based factorization model in Equation 2.13 is given by

$$\mathcal{D}(\theta,\mathbf{R}) = \sum_{u\in\mathcal{U}}\sum_{v\in\mathcal{U}}\left(\mathbf{S}^{(1,1)}_{uv} - sim(u,v)\right)^2 + \sum_{j\in\mathcal{I}}\sum_{i\in\mathcal{I}}\left(\mathbf{S}^{(2,3)}_{ji} - sim(j,i)\right)^2.$$

However, $sim(j,i)$ and $sim(u,v)$ cannot be chosen arbitrarily. Verstrepen and Goethals [104] show that they need to satisfy certain constraints in order to render a well founded unification. Consequently, they propose KUNN, which corresponds to the following similarity definitions that satisfy the necessary constraints:

$$sim(u,v) = \sum_{j\in\mathcal{I}}\frac{\mathbf{R}_{uj}\mathbf{R}_{vj}}{\sqrt{c(u)\cdot c(v)\cdot c(j)}}$$
$$sim(i,j) = \sum_{v\in\mathcal{U}}\frac{\mathbf{R}_{vi}\mathbf{R}_{vj}}{\sqrt{c(i)\cdot c(j)\cdot c(v)}}.$$

In Chapter 3, we discuss these similarity definitions in detail.

Higher Order Neighborhood-based

A fourth set of analytically solvable deviation functions is tailored to the higher order itemset-based neighborhood factorization model of Equation 2.19:

$$\mathbf{S} = \mathbf{X}\mathbf{S}^{(1,2)}.$$

In this case, the deviation function is given by

$$\mathcal{D}(\theta, \mathbf{R}) = \sum_{j \in \mathcal{S}} \sum_{i \in \mathcal{I}} \left(\mathbf{S}^{(1,2)}_{ji} - sim(j,i) \right)^2.$$

with $\mathcal{S} \subseteq 2^{\mathcal{I}}$ the selected itemsets considered in the factorization model.

Deshpande and Karypis [19] propose to define $sim(j,i)$ similar as for the pairwise interactions (Eq. 2.52). Alternatively, others [58, 79] proposed

$$sim(j,i) = sim'(j,i) \cdot \max\left(0, c(j \cup \{i\}) - f\right),$$

with f a hyperparameter.

Lin et al. [53] proposed yet another alternative:

$$sim(j,i) = sim'(j,i) \cdot |top\text{-}k_c(i) \cap \{j\}| \cdot \max\left(0, condProb(j,i) - c\right),$$

with $top\text{-}k_c(i)$ the set of items l that correspond to the k highest values $c(i,l)$, k a hyperparameter, *condProb* the conditional probability similarity according to Equation 2.50 and c a hyperparameter. Furthermore, they define

$$sim'(j,i) = \frac{\left(\sum_{v \in \mathcal{U}} \mathbf{X}_{vi}\mathbf{X}_{vj}\right)^2}{c(j)}. \tag{2.57}$$

A fifth and final set of analytically solvable deviation functions is tailored to the higher order userset-based neighborhood factorization model of Equation 2.20:

$$\mathbf{S} = \mathbf{S}^{(1,1)}\mathbf{Y}.$$

In this case, the deviation function is given by

$$\mathcal{D}(\theta, \mathbf{R}) = \sum_{v \in \mathcal{S}} \sum_{u \in \mathcal{U}} \left(\mathbf{S}^{(1,1)}_{uv} - sim(v,u) \right)^2.$$

with $\mathcal{S} \subseteq 2^{\mathcal{U}}$ the selected usersets considered in the factorization model. Lin et al. [53] proposed to define

$$sim'(v,u) = \frac{\left(\sum_{j \in \mathcal{I}} \mathbf{Y}_{uj}\mathbf{Y}_{vj}\right)^2}{c(v)}. \tag{2.58}$$

Alternatively, Symeonidis et al. [92] propose

$$sim(v,u) = \frac{\sum_{j \in \mathcal{I}} \mathbf{Y}_{uj}\mathbf{Y}_{vj}}{c(v)} \cdot |v| \cdot |top\text{-}k(u)_{cp} \cap \{v\}|,$$

with $top\text{-}k(u)_{cp}$ the set of usersets w that correspond to the k highest values $\frac{\sum_{j \in \mathcal{I}} \mathbf{Y}_{uj}\mathbf{Y}_{wj}}{c(w)}$.

2.4 Minimization Algorithms

Efficiently computing the parameters that minimize a deviation function is often non trivial. Furthermore, there is a big difference between minimizing convex and non-convex deviation functions.

2.4.1 Non-convex Minimization

The two most popular families of minimization algorithms for non-convex deviation functions of collaborative filtering algorithms are *gradient descent* and *alternating least squares*. We discuss both in this section. Furthermore, we also briefly discuss a few other interesting approaches.

Gradient Descent

For deviation functions that assume preferences are missing at random, and consequently consider only the known preferences [83], gradient descent (GD) is generally the numerical optimization algorithm of choice. In GD, the parameters θ are randomly initialized. Then, they are iteratively updated in the direction that reduces $\mathcal{D}(\theta, \mathbf{R})$:

$$\theta^{k+1} = \theta^k - \eta \nabla \mathcal{D}(\theta, \mathbf{R}),$$

with η a hyperparameter called the learning rate. The update step is larger if the absolute value of the gradient $\nabla \mathcal{D}(\theta, \mathbf{R})$ is larger. A version of GD that converges faster is Stochastic Gradient Descent (SGD). SGD uses the fact that $\nabla \mathcal{D}(\theta, \mathbf{R}) = \sum_{t=1}^{T} \nabla \mathcal{D}_t(\mathbf{S}, \mathbf{R})$, with T the number of terms in $\mathcal{D}(\mathbf{S}, \mathbf{R})$. Now, instead of computing $\nabla \mathcal{D}(\theta, \mathbf{R})$ in every iteration, only one term t is randomly sampled (with replacement) and the parameters θ are updated as

$$\theta^{k+1} = \theta^k - \eta \nabla \mathcal{D}_t(\mathbf{S}, \mathbf{R}).$$

Typically, a convergence criterium of choice is only reached after every term t is sampled multiple times on average.

However, when the deviation function assumes the missing feedback is missing not at random, the summation over the known preferences, $\sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} \mathbf{R}_{ui}$, is replaced by a summation over all user item pairs, $\sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}}$, and SGD needs to visit approximately 1000 times more terms. This makes the algorithm less attractive for deviation functions that assume the missing feedback is missing not at random.

To mitigate the large number of terms in the gradient, several authors propose to sample the terms in the gradient not uniformly but proportional to their impact on the parameters [119, 74, 120]. These approaches have not only been proven to speedup convergence, but also to improve the quality of the resulting parameters. Weston et al., on the other hand, sample for every known preference i, a number of non preferred items j until they encounter one for which $\mathbf{S}_{uj} + 1 > \mathbf{S}_{ui}$, i.e. it violates the hinge-loss approximation of the ground truth ranking. In this way, they ensure that every update significantly changes the parameters θ [110].

Alternating Least Squares

If the deviation function allows it, alternating least squares (ALS) becomes an interesting alternative to SGD when preferences are assumed to be missing not at

random [49, 38]. In this respect, the deviation functions of Equations 2.24, 2.29, 2.30, and 2.47 are, amongst others, appealing because they can be minimized with a variant of the alternating least squares (ALS) method. Take for example the deviation function from Equation 2.24:

$$\mathcal{D}(\theta, \mathbf{R}) = \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} \mathbf{W}_{ui} (\mathbf{R}_{ui} - \mathbf{S}_{ui})^2 + \sum_{t=1}^{T} \sum_{f=1}^{F} \lambda_{tf} ||\mathbf{S}^{(t,f)}||_F^2,$$

combined with the basic two-factor factorization model from Equation 2.6:

$$\mathbf{S} = \mathbf{S}^{(1,1)} \mathbf{S}^{(1,2)}.$$

As most deviation functions, this deviation function is non-convex in the parameters θ and has therefore multiple local optima. However, if one temporarily fixes the parameters in $\mathbf{S}^{(1,1)}$, it becomes convex in $\mathbf{S}^{(1,2)}$ and we can analytically find updated values for $\mathbf{S}^{(1,2)}$ that minimize this convex function and are therefore guaranteed to reduce $\mathcal{D}(\theta, \mathbf{R})$. Subsequently, one can temporarily fix the parameters in $\mathbf{S}^{(1,2)}$ and in the same way compute updated values for $\mathbf{S}^{(1,1)}$ that are also guaranteed to reduce $\mathcal{D}(\theta, \mathbf{R})$. One can keep alternating between fixing $\mathbf{S}^{(1,1)}$ and $\mathbf{S}^{(1,2)}$ until a convergence criterium of choice is met. Hu et al. [38], Pan et al. [65] and Pan and Scholz [64] give detailed descriptions of different ALS variations. The version by Hu et al. contains optimizations for the case in which missing preferences are uniformly weighted. Pan and Scholz [64] describe optimizations that apply to a wider range of weighting schemes. Finally, Pilaszy et al. propose to further speed-up the computation by only approximately solving each convex ALS-step [72].

Additionally, ALS has the advantages that it does not require the tuning of a learning rate, that it can benefit from linear algebra packages such as Intel MKL, and that it needs relatively few iterations to converge. Furthermore, when the basic two-factor factorization of Equation 2.6 is used, every row of $\mathbf{S}^{(1,1)}$ and every column of $\mathbf{S}^{(1,2)}$ can be updated independent of all other rows or columns respectively, which makes it fairly easy to massively parallelize the computation of the factor matrices [121].

Bagging

The maximum margin based deviation function in Equation 2.32 cannot be solved with ALS because it contains the hinge loss. Rennie and Srebro propose a conjugate gradients method for minimizing this function [76]. However, this method suffers from similar problems as SGD, related to the high number of terms in the loss function. Therefore, Pan and Scholz [64] propose a bagging approach. The essence of their bagging approach is that they do not explicitly weight every user-item pair for which $\mathbf{R}_{ui} = 0$, but sample from all these pairs instead. They create multiple samples, and compute multiple different solutions $\widetilde{\mathbf{S}}$ corresponding to their samples. These computations are also performed with the conjugate gradients method. They are, however, much less intensive since they only consider a small sample of the many user-item pairs for which $\mathbf{R}_{ui} = 0$. The different solutions $\widetilde{\mathbf{S}}$ are finally aggregated by simply taking their average.

Coordinate Descent

When an item-based neighborhood model is used in combination with a squared error-based deviation function, the user factors are fixed by definition, and the problem resembles a single ALS step. However imposing the constraints in Equation 2.10 complicates the minimization [60]. Therefore, Ning and Karypis adopt cyclic coordinate descent and soft thresholding [25] for SLIM.

2.4.2 Convex Minimization

Aiolli [2] proposed the convex deviation function in Equation 2.48 and indicates that it can be solved with any algorithm for convex optimization.

Also the analytically solvable deviation functions are convex. Moreover, minimizing them is equivalent to computing all the similarities involved in the model. Most works assume a brute force computation of the similarities. However, Verstrepen and Goethals [103], recently proposed two methods that are easily an order of magnitude faster than the brute force computation. In Chapter 5, we discuss these methods in detail.

2.5 Usability Of Rating Based Methods

Interest in collaborative filtering on binary, positive-only data only recently increased. The majority of the existing collaborative filtering research assumes rating data. In this case, the feedback of user u about item i, i.e. $\mathbf{R}_{ui}$, is an integer between B_l and B_h, with B_l and B_h the most negative and positive feedback respectively. The most typical example of rating data was provided in the context of the Netflix Price with $B_l = 1$ and $B_h = 5$.

Technically, our case of binary, positive-only data is just a special case of rating data with $B_l = B_h = 1$. However, collaborative filtering methods for rating data are in general built on the implicit assumption that $B_l < B_h$, i.e. that both positive and negative feedback is available. Since this negative feedback is not available in our problem setting, it is not surprising that, in general, methods for rating data generate poor or even nonsensical results [38, 65, 87].

k-NN methods for rating data, for example, often use the Pearson correlation coefficient as a similarity measure. The Pearson correlation coefficient between users u and v is given by

$$pcc(u,v) = \frac{\sum\limits_{\mathbf{R}_{uj},\mathbf{R}_{vj}>0} (\mathbf{R}_{uj} - \overline{\mathbf{R}}_u)(\mathbf{R}_{vj} - \overline{\mathbf{R}}_v)}{\sqrt{\sum\limits_{\mathbf{R}_{uj},\mathbf{R}_{vj}>0} (\mathbf{R}_{uj} - \overline{\mathbf{R}}_u)^2} \sqrt{\sum\limits_{\mathbf{R}_{uj},\mathbf{R}_{vj}>0} (\mathbf{R}_{vj} - \overline{\mathbf{R}}_v)^2}},$$

with $\overline{\mathbf{R}}_u$ and $\overline{\mathbf{R}}_v$ the average rating of u and v respectively. In our setting, with binary, positive-only data however, $\mathbf{R}_{uj}$ and $\mathbf{R}_{vj}$ are by definition always one or zero. Consequently, $\overline{\mathbf{R}}_u$ and $\overline{\mathbf{R}}_v$ are always one. Therefore, the Pearson correlation is always zero or undefined (zero divided by zero), making it a useless similarity measure for binary, positive-only data. Even if we would hack it by omitting the terms for mean centering, $-\overline{\mathbf{R}}_u$ and $-\overline{\mathbf{R}}_v$, it is still useless since it would always be equal to either one or zero.

Furthermore, when computing the score of user u for item i, user(item)-based k-NN methods for rating data typically find the k users (items) that are most similar to u (i) *and that have rated i (have been rated by u)* [20, 41]. On binary, positive-only data, this approach results in the nonsensical result that $\mathbf{S}_{ui} = 1$ for every (u,i)-pair.

Also the matrix factorization methods for rating data are in general not applicable to binary, positive-only data. Take for example a basic loss function for matrix factorization on rating data:

$$\min_{\mathbf{S}^{(1)},\mathbf{S}^{(2)}} \sum_{\mathbf{R}_{ui}>0} \left(\mathbf{R}_{ui} - \mathbf{S}^{(1)}_{u\cdot}\mathbf{S}^{(2)}_{\cdot i}\right)^2 + \lambda\left(||\mathbf{S}^{(1)}_{u\cdot}||_F^2 + ||\mathbf{S}^{(2)}_{\cdot i}||_F^2\right),$$

which for binary, positive-only data simplifies to

$$\min_{\mathbf{S}^{(1)},\mathbf{S}^{(2)}} \sum_{\mathbf{R}_{ui}=1} \left(1 - \mathbf{S}^{(1)}_{u\cdot}\mathbf{S}^{(2)}_{\cdot i}\right)^2 + \lambda\left(||\mathbf{S}^{(1)}_{u\cdot}||_F^2 + ||\mathbf{S}^{(2)}_{\cdot i}||_F^2\right).$$

The squared error term of this loss function is minimized when the rows and columns of $\mathbf{S}^{(1)}$ and $\mathbf{S}^{(2)}$ respectively are all the same unit vector. This is obviously a nonsensical solution.

The matrix factorization method for rating data that uses singular value decomposition to factorize $\mathbf{R}$ also considers the entries where $\mathbf{R}_{ui} = 0$ and does not suffer from the above problem [80, 17]. Although this method does not result in nonsensical results, the performance has been shown inferior to methods specialized for binary, positive-only data [60, 87, 88].

In summary, although we cannot exclude the possibility that there exists a method for rating data that does perform well on binary, positive-only data, in general this is clearly not the case.

2.6 Conclusions

We have presented a comprehensive survey of collaborative filtering methods for binary, positive-only data. The backbone of our survey is an innovative, unified matrix factorization perspective on collaborative filtering methods, also those that are typically not associated with matrix factorization models such as nearest neighbors methods and association rule mining-based methods. From this perspective, a collaborative filtering algorithm consists of three building blocks: a matrix factorization model, a deviation function and a numerical minimization algorithm. By comparing methods along these three dimensions, we were able to highlight surprising commonalities and key differences.

An interesting direction for future work, is to survey certain aspects that were not included in the scope of this survey. Examples are surveying the different strategies to deal with cold-start problems that are applicable to binary, positive-only data; and comparing the applicability of models and deviation functions for recomputation of models in real time upon receiving novel feedback.

CHAPTER 3

Unifying Nearest Neighbors Collaborative Filtering

An important class of methods for collaborative filtering with binary, positive-only data are the nearest neighbors methods, typically divided into user-based and item-based methods. In this chapter[1]*, we introduce a reformulation that unifies user- and item-based nearest neighbors methods; and use this reformulation to propose a novel method that incorporates the best of both worlds and outperforms state-of-the-art methods. Additionally, we propose a method for naturally explaining the recommendations made by our method and show that this method is also applicable to existing user-based nearest neighbors methods, which were believed to have no natural explanation.*

[1] This chapter is based on work published in RecSys 2014 as "Unifying Nearest Neighbors Collaborative Filtering" by Koen Verstrepen and Bart Goethals [102].

3.1 Introduction

One group of methods for collaborative filtering with binary, positive-only data, are the nearest neighbors methods with analytically computable similarities (Sec. 2.3.11). Sarwar et al. [79] proposed the well known user-based method that uses cosine similarity and Deshpande et al. [19] proposed several widely used item-based methods.

This work builds upon the different item-based methods by Deshpande et al. [19] and the user-based method by Sarwar et al. [79] that uses cosine similarity. We introduce a reformulation of these methods that unifies their existing formulations. From this reformulation, it becomes clear that the existing user- and item-based methods unnecessarily discard important parts of the available information. Therefore, we propose a novel method that combines both user- and item-based information. Hence, this method is neither user-based nor item-based, but *nearest-neighbors-based*. Our experiments show that our method not only outperforms the individual nearest neighbors methods but also state-of-the-art matrix factorization methods.

Furthermore, it is well accepted that every recommendation should come with a short explanation to why it is recommended [38, 35]. Good explanations help users to put the recommendations in the right perspective [95]. Typically, item-based nearest neighbors methods are considered to be superior for this task [20, 38].

Thanks to our reformulation however, we are able to challenge this belief and show that also other nearest neighbors methods have a natural explanation.

The main contributions of this work are:

- We propose a reformulation that unifies user-and item-based nearest neighbors methods (Sec. 3.3) [79, 19].
- We propose KUNN, a novel method for collaborative filtering with binary, positive-only data that is grounded in our unifying reformulation (Sec. 3.4).
- We extensively evaluate our method on real life data and show that it outperforms state-of-the-art methods (Sec. 3.5).
- We propose a method that naturally explains the recommendations by our novel method and user-based methods (Sec. 3.6)

3.2 Preliminaries

We briefly repeat the relevant notations that were introduced in Section 1.3.

Let $\mathcal{U}$ be a set of users and $\mathcal{I}$ a set of items. We are given a matrix with training data $\mathbf{R} \in \{0,1\}^{|\mathcal{U}|\times|\mathcal{I}|}$. $\mathbf{R}_{ui} = 1$ indicates that there is a known preference of user $u \in \mathcal{U}$ for item $i \in \mathcal{I}$. $\mathbf{R}_{ui} = 0$ indicates that there is no such information.

Furthermore, $c(x)$ gives the count of x, meaning

$$c(x) = \begin{cases} \sum_{i\in\mathcal{I}} R_{xi} & \text{if } x \in \mathcal{U} \\ \sum_{u\in\mathcal{U}} R_{ux} & \text{if } x \in \mathcal{I}. \end{cases}$$

The goal of binary, postive-only collaborative filtering is to rank all user-item-pairs (u,i) for which $\mathbf{R}_{ui} = 0$ according to the likelihood of u preferring i. This

is done by computing for every user-item-pair (u,i) a score $s(u,i)$, by which the user-item-pairs are then ranked.

Finally, we denote the $k_{\mathcal{U}}$ ($k_{\mathcal{I}}$) nearest neighbors of a user u (an item i) by $KNN(u)$ ($KNN(i)$).

3.3 Unifying Nearest Neighbors

In this section we propose a novel formulation of nearest neighbors collaborative filtering that unifies the known formulations of the user-based method by Sarwar et al. [79] and the different item-based methods by Deshpande et al. [19]. This formulation is given by

$$s(u,i) = \sum_{v \in \mathcal{U}} \sum_{j \in \mathcal{I}} (\mathcal{L} \cdot \mathcal{N} \cdot \mathcal{G} \cdot \mathcal{S}) \big((u,i),(v,j)\big). \tag{3.1}$$

The score $s(u,i)$, that represents the likelihood that a user u prefers an item i, is computed as a weighted sum over all possible user-item pairs (v,j). The weight of every (v,j)-pair with respect to the pair (u,i) is a multiplication of four functions: the local function $\mathcal{L}\big((u,i),(v,j)\big)$, the neighborhood function $\mathcal{N}\big((u,i),(v,j)\big)$, the global function $\mathcal{G}\big((u,i),(v,j)\big)$ and the rescaling function $\mathcal{S}\big((u,i),(v,j)\big)$.

Before giving a general definition of $\mathcal{L},\mathcal{N},\mathcal{G}$ and $\mathcal{S}$, we first discuss the reformulation of the user-based method by Sarwar et al. [79] and the different item-based methods by Deshpande et al. [19].

3.3.1 Item-Based

Deshpande et al. [19] proposed the now widely used class of item-based nearest neighbors methods. We discuss the variation that uses cosine similarity (Eq. 2.49) without similarity normalization (*cosine, SNorm-*) because of its clean formulation. The analysis for the other variations is analogous.

This method first finds the neighborhood $KNN(j)$ for every preferred item j ($\mathbf{R}_{uj} = 1$) by using the cosine similarity $cos(j,i)$. Next, every preferred item independently increases the score for its $k_{\mathcal{I}}$ most similar items $i \in KNN(j)$ with the similarity value $cos(j,i)$. Thus, the score of a candidate recommendation i for user u is given by [19]:

$$s(u,i) = \sum_{\substack{j \in \mathcal{I} \\ \mathbf{R}_{uj}=1}} cos(j,i) \cdot |KNN(j) \cap \{i\}|. \tag{3.2}$$

We reformulate this method by substituting $cos(j,i)$ by its definition (Eq. 2.49) and regrouping the terms. This gives

$$s(u,i) = \sum_{v \in \mathcal{U}} \sum_{j \in \mathcal{I}} \mathbf{R}_{uj}\mathbf{R}_{vj}\mathbf{R}_{vi} \cdot |KNN(j) \cap \{i\}| \cdot \frac{1}{\sqrt{c(j)c(i)}}.$$

This particular formulation now nicely fits our Equation 3.1: a weighted sum over all possible user-item pairs (v,j) in which the weight of every (v,j)-pair with respect to the pair (u,i) is determined by the functions $\mathcal{L},\mathcal{N},\mathcal{G}$ and $\mathcal{S}$.

The local function $\mathcal{L}$ selects the pairs (v, j) based on their direct relation to the pair (u, i). For the item-based method, it is given by

$$\mathcal{L}\big((u,i),(v,j)\big) = \mathbf{R}_{uj}\mathbf{R}_{vj}\mathbf{R}_{vi}.$$

It thus only selects those pairs (v, j) such that v and u share a preference for j and both i and j are preferred by v.

Next, the neighborhood function $\mathcal{N}$ further selects the pairs (v, j) based on their neighborhood relations to the pair (u, i). For the item-based method, it is given by

$$\mathcal{N}\big((u,i),(v,j)\big) = |KNN(j) \cap \{i\}|.$$

Thus, only those pairs (v, j) for which i is in the neighborhood of j are selected. Notice that the selection of the pair (v, j) by $\mathcal{N}$ is independent of v. As such, the item-based method ignores half of the neighborhood information about the pair (v, j).

Then, the global function $\mathcal{G}$ weighs the selected pairs (v, j) based on global statistics of the items i and j. For the item-based method, it is given by

$$\mathcal{G}\big((u,i),(v,j)\big) = 1/\sqrt{c(i)c(j)}.$$

It thus gives lower weights to those pairs (v, j) for which j has a higher count. Intuitively, if j is more popular, the evidence related to the pair (v, j) is considered less informative. Similarly, if i is more popular, all evidence with respect to $s(u, i)$ is considered less informative. Notice that, in this case, also the weight of the pair (v, j), as determined by $\mathcal{G}$, is independent of v. As such, the item-based method ignores $c(v)$ and $c(u)$, the global information about v and u. Notice furthermore that $\mathcal{G}$ is also used in the definition of the cosine similarity measure (Eq. 2.49), used to determine $KNN(j)$.

As in this case no rescaling is applied, $\mathcal{S}$ is given by

$$\mathcal{S}\big((u,i),(v,j)\big) = 1.$$

3.3.2 User-Based

For a given user u, the user-based nearest neighbors method by Sarwar et al. [79] first finds the neighborhood $KNN(u)$ using the cosine similarity (Eq. 2.56). Next, each neighboring user $v \in KNN(u)$ increases the score of a candidate recommendation i, if i is preferred by v. Thus, the score of a candidate recommendation i for user u is given by [79]:

$$s(u,i) = \frac{1}{|KNN(u)|} \sum_{v \in KNN(u)} \mathbf{R}_{vi}. \tag{3.3}$$

Multiplying the above equation with the constant $|\mathcal{I}| = \sum_{j \in \mathcal{I}} 1$ does not change the ordering of the pairs (u, i) and allows us to rewrite it as

$$s(u,i) = \sum_{v \in \mathcal{U}} \sum_{j \in \mathcal{I}} \mathbf{R}_{vi} \cdot |KNN(u) \cap \{v\}| \cdot \frac{1}{|KNN(u)|}.$$

Hence we have reformulated also the user-based method as a weighted sum over all possible user-item pairs (v, j) in which the weight of a (v, j)-pair with respect to the pair (u, i) is determined by the functions $\mathcal{L}, \mathcal{N}, \mathcal{G}$ and $\mathcal{S}$ (Eq. 3.1).

The local function $\mathcal{L}$, which selects the pairs (v, j) based on their direct relation to the pair (u, i), is for the user-based method given by

$$\mathcal{L}\big((u,i),(v,j)\big) = \mathbf{R}_{vi}.$$

It thus selects those pairs (v, j) such that v prefers i. Unlike the item-based method, it does not consider the information $\mathbf{R}_{uj}$ and $\mathbf{R}_{vj}$ to discriminate between different (v, j)-pairs. Hence, the selection of the pair (v, j) by $\mathcal{L}$ is independent of j. As such, the user-based method ignores local information related to j when weighing the pair (v, j).

Next, the neighborhood function $\mathcal{N}$ further selects the pairs (v, j) based on their neighborhood relations to the pair (u, i). For the user-based method, it is given by

$$\mathcal{N}\big((u,i),(v,j)\big) = |KNN(u) \cap \{v\}|.$$

Thus, only those pairs (v, j) for which v is in the neighborhood of u are selected. Notice that the selection of the pair (v, j) by $\mathcal{N}$ is independent of j. As such, the user-based method ignores half of the neighborhood information about the pair (v, j).

Furthermore, since this method does not weight the pairs (v, j) with any global statistic of u, i, v or j, the global function for the user-based method is given by

$$\mathcal{G}\big((u,i),(v,j)\big) = 1.$$

Finally, for the user-based method, the rescaling function $\mathcal{S}$ rescales the weight of the pairs (v, j) with the size of the neighborhood of u and is therefore given by

$$\mathcal{S}\big((u,i),(v,j)\big) = |KNN(u)|.$$

3.3.3 Generalization

Now we generalize the definitions of the four functions $\mathcal{L}, \mathcal{N}, \mathcal{G}$ and $\mathcal{S}$ such that our formulation covers the most well known user- and item-based methods. Table 3.1 gives an overview of how these functions are defined for both the existing methods and our novel method, which we will propose in the next section.

First, the local function $\mathcal{L}$ selects the pairs (v, j) depending on the direct relations $\mathbf{R}_{ui}$, $\mathbf{R}_{vj}$ and $\mathbf{R}_{vi}$ between (v, j) and (u, i). The user-based method (Sec. 3.3.2) considers only $\mathbf{R}_{vi}$ and ignores the other information. The item-based method (Sec. 3.3.1) on the other hand, combines all three pieces of direct information in the multiplication $\mathbf{R}_{ui}\mathbf{R}_{vj}\mathbf{R}_{vi}$, and thus selects those pairs (v, j) such that v and u share a preference for j and both i and j are preferred by v. Essentially, any combination of these direct relationships between u, i, v and j is possible.

Secondly, the neighborhood function $\mathcal{N}$ weighs the pairs (v, j) depending on the neighborhoods $KNN(u)$, $KNN(v)$, $KNN(i)$ and $KNN(j)$. Existing methods for collaborative filtering consider only one of the four neighborhoods, as shown in Sections 3.3.2 and 3.3.1. Consequently, the weighing function $\mathcal{N}$ reduces to a selection function for these methods . However, any function of these four neighborhoods can be used. For example, in our novel method KUNN, which we will propose in Section 3.4, we use

$$\mathcal{N}\big((u,i),(v,j)\big) = |KNN(u) \cap \{v\}| + |KNN(i) \cap \{j\}|.$$

Table 3.1: Function definitions of the unifying formulation for selected methods. Our novel method is bold faced and marked with a ⋆.

method	$\mathcal{L}\big((u,i),(v,j)\big)$	$\mathcal{N}\big((u,i),(v,j)\big)$	$\mathcal{S}\big((u,i),(v,j)\big)$	$\mathbf{W}\big((u,i),(v,j)\big)$	p_g	$k_{\mathcal{U}}$	$k_{\mathcal{I}}$
user-based, cosine [79]	$\mathbf{R}_{vi}$	$\lvert KNN(u)\cap\{v\}\rvert$	$\mathcal{S}_u(u)$	$\frac{1}{\sqrt{c(u)c(v)}}$	0	$\leq \lvert\mathcal{U}\rvert$	0
item-based, cosine, SNorm- [19]	$\mathbf{R}_{uj}\mathbf{R}_{vj}\mathbf{R}_{vi}$	$\lvert KNN(j)\cap\{i\}\rvert$	1	$\frac{1}{\sqrt{c(i)c(j)}}$	1	0	$\leq \lvert\mathcal{I}\rvert$
item-based, cosine, SNorm+ [19]	$\mathbf{R}_{uj}\mathbf{R}_{vj}\mathbf{R}_{vi}$	$\lvert KNN(j)\cap\{i\}\rvert$	$\mathcal{S}_j(j)$	$\frac{1}{\sqrt{c(i)c(j)}}$	1	0	$\leq \lvert\mathcal{I}\rvert$
⋆ **KUNN**	$\mathbf{R}_{uj}\mathbf{R}_{vj}\mathbf{R}_{vi}$	$\lvert KNN(u)\cap\{v\}\rvert$ $+\lvert KNN(i)\cap\{j\}\rvert$	1	$\frac{1}{\sqrt{c(u)c(j)c(v)c(i)}}$	1	$\leq \lvert\mathcal{U}\rvert$	$\leq \lvert\mathcal{I}\rvert$

Both the user- and the item-based method discussed in the previous two sections used the cosine similarity measure to compute $KNN(x)$ of an object x. Our formulation however, covers a broader range of similarity measures. Let $p_u, p_i, p_v, p_j \in \mathbb{R}$. For users $u, v \in \mathcal{U}$, we define the similarity measure to compute $KNN(u)$ as

$$sim(u,v) = \sum_{r_\mathcal{I} \in \mathcal{I}} \mathbf{R}_{ur_\mathcal{I}} \mathbf{R}_{vr_\mathcal{I}} \cdot c(u)^{p_u} c(v)^{p_v} c(r_\mathcal{I})^{p_j}. \tag{3.4}$$

Similarly, for items $i, j \in \mathcal{I}$, we define the similarity measure to compute $KNN(i)$ as

$$sim(i,j) = \sum_{r_\mathcal{U} \in \mathcal{U}} \mathbf{R}_{r_\mathcal{U} i} \mathbf{R}_{r_\mathcal{U} j} \cdot c(i)^{p_i} c(r_\mathcal{U})^{p_v} c(j)^{p_j}. \tag{3.5}$$

Notice that, in our formulation, the similarity between users (Eq. 3.4) and the similarity between items (Eq. 3.5) share the parameters p_v and p_j. Thus, choosing the user similarity limits the possibilities for choosing the item similarity and vice versa.

Thirdly, the global function $\mathcal{G}$ uses the global statistics $c(u)$, $c(i)$, $c(v)$ and $c(j)$ to weigh the pairs (v, j) with respect to the pair (u, i). It is given by

$$\mathcal{G}\big((u,i),(v,j)\big) = \Big(c(u)^{p_u} c(i)^{p_i} c(v)^{p_v} c(j)^{p_j}\Big)^{p_g}, \tag{3.6}$$

with $p_g \in \{0,1\}$ and p_u, p_i, p_v, p_j the same parameters as in Equations 3.4 and 3.5. Typically, these parameters are negative or zero. In that case, users u, v and items i, j with higher counts reduce the weight of the (v, j)-pairs with respect to (u, i). Intuitively, a more popular item is considered less informative for determining a taste, since this item is more likely preferred by diverse users. Similarly, a user that prefers many items is considered less informative for determining a taste, since this user's preferences are more likely to cover diverse items.

Notice that $\mathcal{G}$, $sim(u, v)$ and $sim(i, j)$ (Eq. 3.6, 3.5 and 3.4) share the factors $c(u)^{p_u}$, $c(i)^{p_i}$, $c(v)^{p_v}$ and $c(j)^{p_j}$. Therefore we introduce the notation

$$\mathbf{W}\big((u,i),(v,j)\big) = c(u)^{p_u} c(i)^{p_i} c(v)^{p_v} c(j)^{p_j},$$

which allows us to rewrite the global function (Eq. 3.6) as

$$\mathcal{G}\big((u,i),(v,j)\big) = \mathbf{W}\big((u,i),(v,j)\big)^{p_g}, \tag{3.7}$$

and the similarities (Eq. 3.4 and 3.5) as

$$sim(u,v) = \sum_{r_\mathcal{I} \in \mathcal{I}} \mathbf{R}_{ur_\mathcal{I}} \mathbf{R}_{vr_\mathcal{I}} \cdot \mathbf{W}\big((u,*),(v,r_\mathcal{I})\big),$$
$$sim(i,j) = \sum_{r_\mathcal{U} \in \mathcal{U}} \mathbf{R}_{r_\mathcal{U} i} \mathbf{R}_{r_\mathcal{U} j} \cdot \mathbf{W}\big((*,i),(r_\mathcal{U},j)\big),$$

with $c(*) = 1$. This definition of $\mathbf{W}$ covers both the user-based method by Sarwar et al. [79] and the different item-based methods by Deshpande et al. [19]. A more general definition of $\mathbf{W}$ would cover a broader range of nearest neighbors methods. We choose this definition over a more general one to emphasize the strong similarity between the latter two methods.

Finally, some methods rescale the weights of the pairs (v, j) with the density of $KNN(u)$, $KNN(i)$, $KNN(v)$ or $KNN(j)$. A neighborhood $KNN(x)$ of x is denser if

the total distance of x to its neighbors is smaller. In other words, if the sum of the similarities of x to its neighbors is higher. Depending on the choice for $KNN(u)$, $KNN(i)$, $KNN(v)$ or $KNN(j)$, the rescaling function is given by one of the four density functions

$$\mathcal{S}_u(u) = 1/\sum_{r_{\mathcal{U}} \in KNN(u)} sim(u, r_{\mathcal{U}})^{p_g},$$
$$\mathcal{S}_i(i) = 1/\sum_{r_{\mathcal{I}} \in KNN(i)} sim(i, r_{\mathcal{I}})^{p_g},$$
$$\mathcal{S}_v(v) = 1/\sum_{r_{\mathcal{U}} \in KNN(v)} sim(r_{\mathcal{U}}, v)^{p_g},$$
$$\mathcal{S}_j(j) = 1/\sum_{r_{\mathcal{I}} \in KNN(j)} sim(r_{\mathcal{I}}, j)^{p_g},$$

with p_g the same parameter as for the global function $\mathcal{G}$ (Eq. 3.7).

For a method that does not apply rescaling,

$$\mathcal{S}\big((u,i),(v,j)\big) = 1$$

3.4 KUNN Unified Nearest Neighbors

Looking at Table 3.1, we observe that the user-based method by Sarwar et al. [79] ignores the information $\mathbf{R}_{uj}$, $\mathbf{R}_{vj}$, $c(i)$, $c(j)$, $KNN(i)$ and $KNN(j)$ for weighing the pairs (v, j) with respect to (u, i). Similarly, all item-based methods by Deshpande et al. [19] ignore the information $c(u)$, $c(v)$, $KNN(u)$ and $KNN(v)$ for weighing the pairs (v, j) with respect to (u, i). Thus, the existing methods ignore an important part of the available information. What is more, the information ignored by item-based methods is disjoint with the information ignored by the user-based method. However, the fact that both user- and item-based methods generate acceptable results [79, 19], indicates that most likely both types of information are useful. Therefore, a novel method that combines both types of information, KUNN[2], potentially leads to improved results. The experiments discussed in Section 3.5 confirm that this is indeed the case. It is not only possible to outperform the individual user- and item-based methods, but also to outperform state-of-the-art matrix factorization methods [75, 38, 65].

The definitions of $\mathcal{L}$, $\mathcal{N}$, $\mathcal{G}$ and $\mathcal{S}$ corresponding to KUNN are given on the last row of Table 3.1.

For KUNN, the local function is given by

$$\mathcal{L}\big((u,i),(v,j)\big) = \mathbf{R}_{uj}\mathbf{R}_{vj}\mathbf{R}_{vi}$$

and thus selects those pairs (v, j) such that v and u share a preference for j and both i and j are preferred by v. As such, KUNN does not discard any information about the direct relation between (v, j) and (u, i).

Next, the neighborhood function for KUNN is given by

$$\mathcal{N}\big((u,i),(v,j)\big) = |KNN(u) \cap \{v\}| + |KNN(i) \cap \{j\}|.$$

[2] KUNN is a recursive acronym for KUNN Unified Nearest Neighbors

It thus selects those pairs (v, j) such that v is in the neighborhood of u or j is in the neighborhood of i. If both conditions are fulfilled, the weight of the pair (v, j) with respect to $s(u, i)$ is doubled. As such, KUNN uses neighborhood information of both v and j to weight (v, j) with respect to (u, i).

Furthermore, for KUNN, $\mathbf{W}$ is given by

$$\mathbf{W}\big((u,i),(v,j)\big) = \frac{1}{\sqrt{c(u)c(i)c(v)c(j)}}.$$

Consequently, the user-similarity results in

$$sim(u,v) = \sum_{r_{\mathcal{I}} \in \mathcal{I}} \frac{\mathbf{R}_{ur_{\mathcal{I}}}\mathbf{R}_{vr_{\mathcal{I}}}}{\sqrt{c(u)c(v)c(r_{\mathcal{I}})}},$$

and the item-similarity:

$$sim(i,j) = \sum_{r_{\mathcal{U}} \in \mathcal{U}} \frac{\mathbf{R}_{r_{\mathcal{U}}i}\mathbf{R}_{r_{\mathcal{U}}j}}{\sqrt{c(i)c(r_{\mathcal{U}})c(j)}}.$$

Intuitively, if u and v share a preference for an item $r_{\mathcal{I}}$, it is only weak evidence of their similarity if $r_{\mathcal{I}}$ is popular and both u and v have many preferences. Similarly, if i and j are both preferred by $r_{\mathcal{U}}$, it is only weak evidence of their similarity if $r_{\mathcal{U}}$ has many preferences and both i and j are popular items.

In addition, the global function for KUNN is given by

$$\mathcal{G}\big((u,i),(v,j)\big) = \mathbf{W}\big((u,i),(v,j)\big)^1 = \frac{1}{\sqrt{c(u)c(i)c(v)c(j)}}.$$

Intuitively, if the counts of u, i, v and j are higher, it is more likely that the direct relation between (v, j) and $(u, i)(\mathcal{L}\big((u,i),(v,j)\big) = 1)$, exists by chance. Therefore, this direct relation is less informative.

Finally, we see no convincing arguments for making KUNN more complex by introducing a rescaling factor. Therefore we define

$$\mathcal{S}\big((u,i),(v,j)\big) = 1.$$

To enhance the intuitive understanding of KUNN, we rewrite Equation 3.1 as

$$s(u,i) = \frac{s_{\mathcal{U}}(u,i) + s_{\mathcal{I}}(u,i)}{\sqrt{c(u)c(i)}}, \tag{3.8}$$

with

$$s_{\mathcal{U}}(u,i) = \sum_{v \in KNN(u)} \mathbf{R}_{vi} \frac{1}{\sqrt{c(v)}} \sum_{\substack{j \in \mathcal{I} \\ \mathbf{R}_{uj}=1 \\ \mathbf{R}_{vj}=1}} \frac{1}{\sqrt{c(j)}}$$

and

$$s_{\mathcal{I}}(u,i) = \sum_{j \in KNN(i)} \mathbf{R}_{uj} \frac{1}{\sqrt{c(j)}} \sum_{\substack{v \in \mathcal{U} \\ \mathbf{R}_{vi}=1 \\ \mathbf{R}_{vj}=1}} \frac{1}{\sqrt{c(v)}}.$$

Thus, we can decompose $s(u,i)$ in a user-based part $s_{\mathcal{U}}(u,i)$ and an item-based part $s_{\mathcal{I}}(u,i)$. Notice that these two parts cannot be reduced to any existing user- or item-based method.

The user-based part $s_{\mathcal{U}}(u,i)$ is a weighted sum over the neighbors of u in which the weight of a neighbor v is proportional to:

- $\mathbf{R}_{vi}$: v has a known preference for i,
- $1/\sqrt{c(v)}$: if v prefers many items, her known preference for i becomes less informative,
- $\sum_{j \in \mathcal{I}, \mathbf{R}_{uj}=1, \mathbf{R}_{vj}=1}$: every preference that v shares with u increases the weight of v for recommending items to u,
- $1/\sqrt{c(j)}$: if v and u share a preference for j, it is less informative if j is a more popular item.

A similar intuition holds for the item-based part $s_{\mathcal{I}}(u,i)$.

Finally, the denominator of Equation 3.8, reduces the strength of the evidence if u prefers many items and i is popular.

3.5 Experimental Evaluation

We experimentally evaluate the accuracy of KUNN on three datasets: the *Movielens*, the *Yahoo!Music*^{user} and the *Yahoo!Music*^{random} datasets [113, 32]. These datasets contain ratings of users for movies and songs respectively. The ratings are on a 1 to 5 scale with 5 expressing the highest preference. We convert these datasets to binary, positive-only datasets. Following Pradel et al. [73], we convert the ratings 4 and 5 to preferences and the ratings 1 and 2 to dislikes. Furthermore, we ignore the ratings 3, effectively converting them to unknowns. As such, we obtain a buffer between preferences and dislikes. Since our setting presumes binary, positive-only data, both the unknowns and the dislikes are represented by zeros in the training data. For evaluating the recommendations however, we are allowed to distinguish between unknowns and dislikes.

In our evaluation we compare *KUNN* with five other methods. As a baseline, we select *pop*, the non-personalized method that ranks all items according to their popularity, i.e. the number of users in the training set that prefer the item. Next, we select the *user-based* nearest neighbors method with cosine similarity by Sarwar et al. [79] and the widely used *item-based* nearest neighbors method with cosine similarity and similarity normalization (*SNorm+*) by Deshpande et al. [19]. We choose this item-based method because it performed well in comparison to other item-based methods [19]. Furthermore, we also compare with *UB+IB*, a linear ensemble that computes a recommendation score as

$$s(u,i) = \lambda s_{UB}(u,i) + (1-\lambda) s_{IB}(u,i),$$

with $s_{UB}(u,i)$ the user-based score from the method by Sarwar et al. and $s_{IB}(u,i)$ the item-based score from the method by Deshpande et al. Finally, we compare with two state-of-the-art matrix factorization methods for collaborative filtering with binary, positive-only data: the *BPRMF* method by Rendle et al. and the *WRMF* method by Hu

Table 3.2: Dataset characteristics after transformation to binary, positive-only data.

				$c(u)$		$\lvert\mathcal{H}_u\rvert$		$\lvert\mathcal{M}_u\rvert$	
Dataset	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{U}_t\rvert$	$\lvert\mathcal{I}\rvert$	mean	std	mean	std	mean	std
Movielens	6040	6037	3706	94.3	105.0	1	0	NA	NA
Yahoo!Music^{user}	15400	13204	1000	7.70	11.57	1	0	NA	NA
Yahoo!Music^{random}	15400	2401	1000	8.70	11.57	1.89	1.23	6.17	2.09

et al. [38]. For *WRMF* and *BPRMF* we used the MyMediaLite implementation [28]. For all other methods, we used our own implementation, which is available at `https://bitbucket.org/KVs/unncf_submit`.

To thoroughly evaluate the performance of KUNN, we use evaluation measures from multiple previous works [19, 75, 73, 79]. The experimental evaluation consists of two experimental setups. In the *user selected* setup (Sec. 3.5.1), the users selected which items they rated. In the *random selected* setup (Sec. 3.5.2), users were asked to rate randomly chosen items.

3.5.1 User Selected Setup

This experimental setup is applicable to the *Movielens* and the *Yahoo!Music*^{user} datasets. In both datasets, users chose themselves which items they rated. Following Deshpande et al. [19] and Rendle et al. [75], one preference of every user is randomly chosen to be the test preference for that user. If a user has only one preference, no test preference is chosen. The remaining preferences are represented as a 1 in the training matrix $\mathbf{R}$. All other entries of $\mathbf{R}$ are zero. We define the hit set $\mathcal{H}_u$ of a user u as the set containing all test preferences of that user. For this setup, this is a singleton, denoted as $\{h_u\}$, or the empty set if no test preference is chosen. Furthermore, we define $\mathcal{U}_t$ as the set of users with a test preference, i.e. $\mathcal{U}_t = \{u \in \mathcal{U} \mid |\mathcal{H}_u| > 0\}$. Table 3.2 summarizes some characteristics of the datasets.

For every user $u \in \mathcal{U}_t$, every method ranks the items $\{i \in \mathcal{I} \mid \mathbf{R}_{ui} = 0\}$ based on $\mathbf{R}$. We denote the rank of the test preference h_u in such a ranking as $r(h_u)$.

Then, every set of rankings is evaluated using three measures. Following Deshpande et al. [19] we use hit rate at 10 and average reciprocal hit rate at 10. In general, 10 can be replaced by any natural number $N \leq |\mathcal{I}|$. We follow Deshpande et al. [19] and choose N=10.

Hit rate at 10 is given by

$$HR@10 = \frac{1}{|\mathcal{U}_t|} \sum_{u \in \mathcal{U}_t} |\mathcal{H}_u \cap top10(u)|,$$

with $top10(u)$ the 10 highest ranked items for user u. Hence, *HR*@10 gives the percentage of test users for which the test preference is in the top 10 recommendations.

Average reciprocal hit rate at 10 is given by

$$ARHR@10 = \frac{1}{|\mathcal{U}_t|} \sum_{u \in \mathcal{U}_t} |\mathcal{H}_u \cap top10(u)| \cdot \frac{1}{r(h_u)}.$$

Unlike hit rate, average reciprocal hit rate takes into account the rank of the test preference in the top 10 of a user.

Following Rendle et al. [75], we also use the AMAN version of area under the curve, which is for this experimental setup given by

$$AUC^{AMAN} = \frac{1}{|\mathcal{U}_t|} \sum_{u \in \mathcal{U}_t} \frac{|\mathcal{I}| - r(h_u)}{|\mathcal{I}| - 1}.$$

AMAN stands for All Missing As Negative, meaning that a missing preference is evaluated as a dislike. Like average reciprocal hit rate, the area under the curve takes into account the rank of the test preference in the recommendation list for a user. However, AUC^{AMAN} decreases slower than $ARHR@10$ when $r(h_u)$ increases.

We repeat all experiments five times, drawing a different random sample of test preferences every time.

3.5.2 Random Selected Setup

The *user selected* experimental setup introduces two biases in the evaluation. Firstly, popular items get more ratings. Secondly, the majority of the ratings is positive. These two biases can have strong influences on the results and are thoroughly discussed by Pradel et al. [73]. The *random selected* test setup avoids these biases.

Following Pradel et al. [73], the training dataset is constructed in the *user selected* way: users chose to rate a number of items they selected themselves. The test dataset however, is the result of a voluntary survey in which random items were presented to the users and a rating was asked. In this way, both the popularity and the positivity bias are avoided.

This experimental setup is applicable to the dataset $Yahoo!Music^{random}$. The training data, $\mathbf{R}$, of this dataset is identical to the full $Yahoo!Music^{user}$ dataset. Additionally, this dataset includes a test dataset in which the rated items were randomly selected. For a given user u, the hit set $\mathcal{H}_u$ contains all preferences of this user which are present in the test dataset. The set of dislikes $\mathcal{M}_u$ contains all dislikes of this user which are present in the test dataset. We define $\mathcal{U}_t = \{u \in \mathcal{U} \mid |\mathcal{H}_u| > 0, |\mathcal{M}_u| > 0\}$, i.e. all users with both preferences and dislikes in the test dataset. Table 3.2 summarizes some characteristics of the dataset.

For every user $u \in \mathcal{U}_t$, every method ranks the items in $\mathcal{H}_u \cup \mathcal{M}_u$ based on the training data $\mathbf{R}$. The rank of an item i in such a ranking is denoted $r(i)$.

Then, following Pradel et al. [73], we evaluate every set of rankings with the AMAU version of area under the curve, which is given by

$$AUC^{AMAU} = \frac{1}{|\mathcal{U}_t|} \sum_{u \in \mathcal{U}_t} AUC^{AMAU}(u),$$

with

$$AUC^{AMAU}(u) = \sum_{h \in \mathcal{H}_u} \frac{|\{m \in \mathcal{M}_u \mid r(m) > r(h)\}|}{|\mathcal{H}_u||\mathcal{M}_u|}.$$

AMAU stands for All Missing As Unknown, meaning that a missing preference does not influence the evaluation. Hence, $AUC^{AMAU}(u)$ measures, for a user u, the fraction of dislikes that is, on average, ranked behind the preferences. A big advantage of

this measure is that it only relies on known preferences and known dislikes. Unlike the other measures, it does not make the bold assumption that items not preferred by u in the past are disliked by u.

Because of the natural split in a test and training dataset, repeating the experiment with different test preferences is not applicable.

3.5.3 Parameter Selection

Every personalization method in the experimental evaluation has at least one parameter. For every experiment we try to find the best set of parameters using grid search. An experiment is defined by (1) the outer training dataset $\mathbf{R}$, (2) the outer hitset $\bigcup_{u \in \mathcal{U}_t} \mathcal{H}_u$, (3) the outer dislikes $\bigcup_{u \in \mathcal{U}_t} \mathcal{M}_u$, (4) the evaluation measure, and (5) the method. Applying grid search, we first choose a finite number of parameter sets. Secondly, from the training data $\mathbf{R}$, we create five inner data splits, defined by $\mathbf{R}^k$, $\bigcup_{u \in \mathcal{U}_t} \mathcal{H}^k_u$, and $\bigcup_{u \in \mathcal{U}_t} \mathcal{M}_u$ for $k \in \{1,2,3,4,5\}$. Then, for every parameter set, we rerun the method on all five inner training datasets $\mathbf{R}^k$ and evaluate them on the corresponding inner test datasets with the chosen evaluation measure. Finally, the parameter set with the best average score over the five inner data splits on a certain evaluation measure, is chosen to compute recommendations for the outer training dataset $\mathbf{R}$ concerning the evaluation with the same evaluation meaure. Notice that, given an outer training dataset, the best parameters with respect to one evaluation measure, can differ from the best parameters with respect to another evaluation measure.

3.5.4 Results and Discussion

Table 3.3 shows the evaluation of the considered methods on different datasets, with different measures. The experiments belonging to the user selected setup were repeated 5 times, drawing a different random sample of test preferences every time. Therefore, we report both the mean and the standard deviation for these experiments. The experiments belonging to the random selected setup use a natural split between the test and training dataset. Therefore, randomizing the test set choice is not applicable and only one value is reported for every experiment. Scripts for automatically repeating all experiments are available at `https://bitbucket.org/KVs/unncf_submit`. The exact paramter combinations explored by the grid search procedure (Sec. 3.5.3) and other details can be inspected in these scripts.

From Table 3.3 we can make several observations. First of all, KUNN outperforms every other method five out of seven times, shares one best performance with WRMF, and is one time outperformed by WRMF.

Secondly, the user-based method clearly outperforms the item-based method on the *Movielens* dataset. On the *Yahoo!Music*^{user} dataset on the other hand, the item-based method clearly outperforms the user-based method. For UB+IB, the grid search procedure (Sec. 3.5.3) is successful in choosing λ such that the best of both methods in the ensemble gets the highest weight. However, UB+IB cannot outperform the best of both individual methods. KUNN, on the other hand, successfully combines the user- and item-based information and consistently outperforms both individual methods.

Table 3.3: Selected methods evaluated with multiple experiments. Our novel method is marked with a ⋆

dataset	measure		⋆ KUNN	WRMF [38, 65]	BPRMF [75]	UB+IB	item-based, cosine, SNorm+ [19]	user-based, cosine [79]	pop
Movielens	*HR*@10	mean	**.217**	**.217**	.174	.209	.165	.209	.063
		std	.002	.001	.004	.002	.002	.002	.002
	ARHR@10	mean	**.093**	.091	.071	.090	.073	.090	.022
		std	.003	.002	.001	.002	.002	.002	.002
	AUC^{AMAN}	mean	.941	**.944**	.934	.927	.916	.927	.865
		std	.001	.001	.002	.001	.001	.001	.001
$Yahoo!Music^{user}$	*HR*@10	mean	**.378**	.338	.285	.365	.364	.316	.187
		std	.009	.007	.009	.003	.012	.012	.004
	ARHR@10	mean	**.163**	.145	.102	.160	.160	.127	.062
		std	.003	.007	.006	.006	.007	.003	.002
	AUC^{AMAN}	mean	**.926**	.910	.891	.910	.911	.910	.841
		std	.002	.002	.003	.003	.003	.003	.001
$Yahoo!Music^{random}$	AUC^{AMAU}		**.788**	.770	.755	.768	.768	.761	.670

Thirdly, the rather disappointing performance of BPRMF stands out. A possible explanation lies in the choice of the parameters. Following Rendle et al. [75], we used grid search (Sec. 3.5.3) to choose the best out of 267 parameter combinations for BPRMF in every experiment. We can however not rule out that there exist parameter combinations outside our 267 possibilities, for which BPRMF performs better. Finding a good set of parameters is harder for BPRMF than for the other methods because BPRMF uses 7 parameters that can all take an infinite number of values. The parameters $k_{\mathcal{U}}$ and $k_{\mathcal{I}}$ of KUNN, on the other hand, are integers within the bounds $[0, |\mathcal{U}|]$ and $[0, |\mathcal{I}|]$ respectively. Therefore, we can find a good set of parameters among only 25 possibilities.

Finally, all personalized methods perform much better than the non personalized baseline *pop*.

3.6 Explainability

The consideration that explanations of item-based methods are superior comes from observing the formulas for computing the recommendation scores [20, 38]. For item-based methods on the one hand, this formula is given by Equation 3.2 in which every term can be attributed to one of the known preferences of the target user. Therefore the known preferences related to the biggest terms can naturally serve as an explanation for the recommendation. For user-based methods on the other hand, the formula is given by Equation 3.3 in which every term can be attributed to one of the collaborative users. This is much less useful because the most similar users give no intuitive explanation for the recommendation as they are probably strangers to the target user and the same for every recommendation. Furthermore, this kind of explanation would also be an invasion on the privacy of these collaborative users.

However, this difference is only artificial. Thanks to our reformulation, we can write both the user-based method by Sarwar et al. [79] (Eq. 3.3) and KUNN as a sum over the known preferences of the target user.

We start with the user-based method (Eq. 3.3). This method can be rewritten as

$$s(u,i) = \frac{1}{|KNN(u)|} \sum_{v \in KNN(u)} \mathbf{R}_{vi} \left(\frac{\sum_{l_1 \in \mathcal{I}} \mathbf{R}_{ul_1} \mathbf{R}_{vl_1}}{\sum_{l_2 \in \mathcal{I}} \mathbf{R}_{ul_2} \mathbf{R}_{vl_2}} \right).$$

Notice that the last factor in the summation is simply 1, but allows us to rewrite the equation as

$$s(u,i) = \frac{1}{|KNN(u)|} \sum_{\substack{l_1 \in \mathcal{I} \\ \mathbf{R}_{ul_1}=1}} \left(\sum_{v \in KNN(u)} \frac{\mathbf{R}_{vl_1} \mathbf{R}_{vi}}{\sum_{l_2 \in \mathcal{I}} \mathbf{R}_{ul_2} \mathbf{R}_{vl_2}} \right),$$

In the above equation, we have written the user based score $s(u,i)$ as a weighted sum over the known preferences of u.

The known preferences l_1 with the biggest weights, serve as a natural explanation for recommending i to u. Hence, we have naturally explained the user-based recommendation of i for u.

Next, we consider KUNN, which can be rewritten as

$$s(u,i) = \frac{1}{\sqrt{c(u)c(i)}} \sum_{\substack{j \in \mathcal{I} \\ \mathbf{R}_{uj}=1}} \left(\frac{1}{\sqrt{c(j)}} \sum_{\substack{v \in \mathcal{U} \\ \mathbf{R}_{vi}=1 \\ \mathbf{R}_{vj}=1}} \frac{\mathcal{N}\big((u,i),(v,j)\big)}{\sqrt{c(v)}} \right),$$

by regrouping Equation 3.1. Thus, also KUNN computes $s(u,i)$ as a weighted sum over the known preferences of u.

Again, the known preferences with the biggest weights serve as a natural explanation for recommending i to u. Hence, we have naturally explained recommendations made by KUNN.

3.7 Related Work

We surveyed methods for collaborative filtering with binary, positive-only data in Chapter 2.

Furthermore, Symeonidis et al. [92] recognized that combining user- and item-based approaches could be beneficial. Additionally, Wang et al. [105] proposed a unification of user- and item-based methods. However, their work presumes rating data.

3.8 Conclusions

We proposed KUNN, a novel method for one class collaborative filtering, a setting that covers many applications.

KUNN originates from a reformulation that unifies user- and item-based nearest neighbors methods. Thanks to this reformulation, it becomes clear that user- and item-based nearest neighbors methods discard important parts of the available information.

KUNN improves upon these existing nearest neighbors methods by actually using more of the available information. Our experimental evaluation shows that KUNN not only outperforms existing nearest neighbors methods, but also state-of-the-art matrix factorization methods.

Finally, we challenged the well accepted belief that item-based methods are superior for explaining the recommendations they produce. Thanks to our reformulation, we were able to show that also recommendations by KUNN and the traditional user-based method come with a natural explanation.

We see research on novel definitions of the functions $\mathcal{L}$, $\mathcal{N}$, $\mathcal{G}$ and $\mathcal{S}$ as the most important direction for future work.

CHAPTER 4

Top-N Recommendation for Shared Accounts

The vast majority of collaborative filtering recommender systems assume that every account in the training data represents a single user. However, multiple users often share a single account. A typical example is a single shopping account for the whole family. Traditional recommender systems fail in this situation. If contextual information is available, context aware recommender systems are the state-of-the-art solution. Yet, often no contextual information is available. Therefore, we introduce in this chapter[1] the challenge of recommending to shared accounts in the absence of contextual information. We propose a solution to this challenge for all cases in which the reference recommender system is an item-based top-N collaborative filtering recommender system, generating recommendations based on binary, positive-only data. We experimentally show the advantages of our proposed solution for tackling the problems that arise from the existence of shared accounts on multiple datasets.

[1] This chapter is based on work published in RecSys 2015 as "Top-N Recommendation for Shared Accounts" by Koen Verstrepen and Bart Goethals [104].

4.1 Introduction

Typical recommender systems assume that every user- account represents a single user. However, multiple users often share a single account. An example is a household in which all people share one video-streaming account, one music-streaming account, one online-shopping account, one loyalty card for a store they make purchases in, etc.

Three problems arise when multiple users share one account. First, the *dominance problem* arises when all recommendations are relevant to only some of the users that share the account and at least one user does not get any relevant recommendation. We say that these few users dominate the account. Consider, for example, a family that often purchases household items. Once in a while they also purchase toys for the children together with the household items. Now, it is likely that all recommendations will be based on the numerous household items and the recommender system will be essentially useless for the children.

Second, the *generality problem* arises when the recommendations are only a little bit relevant to all users in the shared account, but are not really appealing to any of them. When the diverse tastes of multiple users are merged into one account, the recommender system is more likely to recommend overly general items that are preferred by most people, regardless their individual tastes.

Third, if the recommender system would be able to generate relevant recommendations for every user in the shared account, how does every user know which recommendation is meant for her? We call this the *presentation problem*.

If contextual information such as time, location, buying intent, item content, session logs, etc. is available, context aware recommender systems are the state-of-the-art solution to split accounts into multiple users and detect the identity of the active user at recommendation time.

However, often no contextual information is available for splitting the accounts. A first example concerns the case of the numerous organizations that simply did not keep records of any contextual information in the past, not even time stamps. A second example are families that shop together in a hypermarket: they have one loyalty card account and bundle their purchases when they visit the store. In this case, the context is exactly the same for every family member and cannot be used to split the family account into its members. Therefore, we introduce the challenge of *top-N recommendation for shared accounts in the absence of contextual information*, in which the above three shared account problems are tackled without using any contextual information.

Despite the significance of *top-N recommendation for shared accounts in the absence of contextual information*, we know of no prior research on tackling all aspects of this challenge. We give a start to filling this gap by proposing a solution for all cases in which the reference recommender system is an item-based top-N collaborative filtering recommender system, generating recommendations based on binary, positive-only feedback (Sec. 2.9). In this way, we cover a large number of applications since item-based top-N collaborative filtering recommender systems are very popular. Multiple authors attribute this popularity to the combination of favorable properties such as simplicity, stability, efficiency, reasonable accuracy, the ability for intuitively explaining their recommendations, and the ability for immediately taking into account newly entered feedback [20, 48, 41, 54].

Central to our approach, we show a property of item-based top-N collaborative filtering recommender systems that allows us to compute a recommendation score in $\mathcal{O}(n \log n)$ instead of exponential time.

The main contributions of this work are:

- We formally introduce the challenge of *top-N recommendation for shared accounts in the absence of contextual information* (Sec. 4.2).
- We propose a solution to this challenge for all cases in which the reference recommender system is an item-based top-N collaborative filtering recommender system, generating recommendations based on binary, positive-only feedback [19] (Sec. 4.5-4.8).
- Most importantly, we show an essential property of item-based top-N collaborative filtering recommender systems that allows us to keep the time complexity of our proposed solution within practically feasible limits (Sec. 4.6).
- We experimentally show on multiple datasets that our proposed solution is able to detect preferences of individual users in shared accounts and has therefore significant advantages for tackling the dominance, generality and presentation problems (Sec. 4.9).

After formalizing the definitions of the challenge (Sec. 4.2) and the reference recommender system (Sec. 4.3) we first give further insight in how the reference recommender system suffers from the shared account problems (Sec. 4.4). Afterwards we sequentially solve the generality problem (Sec. 4.5), the dominance problem (Sec. 4.7) and the presentation problem (Sec. 4.8). Furthermore, we inserted a section on the efficient computation of our solution to the generality problem (Sec. 4.6). Finally, we discuss the experimental evaluation of our proposed solution (Sec. 4.9).

4.2 Problem Definition

We adhere to the notation in which $\mathcal{U}$ is the set of users and $\mathcal{I}$ is the set of items. Furthermore, let $\mathcal{A}$ be the set of accounts, let $U(a) \subseteq \mathcal{U}$ be the set of users that share account a, i.e. the userset of account a, and let $a(u) \in \mathcal{A}$ be the account that user u belongs to. Notice that in this problem setting every user belongs to exactly one account.

First, consider the user-rating-matrix $\mathbf{T} \in \{0,1\}^{|\mathcal{U}| \times |\mathcal{I}|}$. $\mathbf{T}_{ui} = 1$ indicates that there exists a preference of user $u \in \mathcal{U}$ for item $i \in \mathcal{I}$. $\mathbf{T}_{ui} = 0$ indicates that there is no such preference.

We are given a reference recommender system R_{ref} that produces the desired recommendations given $\mathbf{T}$. Consequently, we say that an item i is **relevant** to a user u if i is in the top-N recommendations for u as computed by the reference recommender system $R_{ref}(\mathbf{T})$ on the user-rating-matrix $\mathbf{T}$.

Unfortunately, in our problem setting $\mathbf{T}$ is unknown. Instead we are given the account-rating-matrix $\mathbf{R} \in \{0,1\}^{|\mathcal{A}| \times |\mathcal{I}|}$. $\mathbf{R}_{ai} = 1$ indicates that there is a known preference of account $a \in \mathcal{A}$ for item $i \in \mathcal{I}$. $\mathbf{R}_{ai} = 0$ indicates that there is no such information.

Now, the challenge of *top-N recommendation for shared accounts in the absence of contextual information* is to devise a shared account recommender system $R_{sa}(\mathbf{R})$ that, based on the account-rating-matrix $\mathbf{R}$, computes for every account a the top N_a recommendations such that:

- Ideally, this top-N_a contains all top-N items for every user in the userset of a, with $N = \frac{N_a}{|U(a)|}$. Practically, the goal is to avoid the dominance and the generality problem by maximizing the number of users with at least one item from its top-N.

- It is clear for a user in the userset of a which items in the top-N_a are meant for her, i.e. the presentation problem gets solved.

Notice that in the above definition, the shared account recommender system does **not** get the number of users sharing every account as an input. Furthermore, no assumption is made about the shared interests of the users sharing an account. They can have totally different interests, partially overlapping interests or fully overlapping interests.

Finally, notice that this problem definition is orthogonal to a typical group recommendation problem [55]. First, in group recommendation, the individual profiles of the users in the shared account are typically known. Here, they are unknown. Second, in group recommendation, it is typically assumed that the recommendations will be consumed by all users in the shared account together. Here, it is assumed that every user in the shared account can identify the recommendations meant for her and consumes these recommendations individually.

4.3 The Reference Recommender System

Typically, recommender systems find the top-N recommendations for a user u by first computing the recommendation scores $s(u,i)$ for every candidate recommendation i and afterwards selecting the N recommendations i for which $s(u,i)$ is the highest.

One of the most popular models for collaborative filtering with binary, positive-only data are the item-based collaborative filtering recommender systems (Eq. 2.9). These item-based recommender systems are rooted in the intuition that good recommendations are similar to the items already preferred by the target user. Thus, for a target user u, this recommender system first finds $KNN(j)$, the k most similar items to j, for every preferred item j ($\mathbf{T}_{uj} = 1$) by using the similarity values $sim(j,i)$. Next, every preferred item independently increases the recommendation score for its k most similar items $i \in KNN(j)$ with the similarity value $sim(j,i)$. Thus, the item-based recommendation score of a candidate recommendation i for user u is given by [19]:

$$\begin{aligned} s_{IB}(u,i) &= s_{IB}(I(u),i) \\ &= \sum_{j \in I(u)} sim(j,i) \cdot |KNN(j) \cap \{i\}|, \end{aligned} \tag{4.1}$$

with $I(u) = \{j \in \mathcal{I} \mid \mathbf{T}_{uj} = 1\}$, the set of items preferred by u.

A typical choice for $sim(j,i)$ is the cosine similarity. Furthermore, Deshpande et al. report that normalizing the cosine similarity scores improves the performance [19]. This comes down to defining $sim(j,i)$ in Equation 4.1 as:

$$sim(j,i) = \frac{cos(j,i)}{\sum_{l \in KNN(j)} cos(j,l)}.$$

We will use this recommender system as the reference recommender system R_{ref}.

4.4 Shared Account Problems of the Reference Recommender System

Simply applying the reference recommender system (Sec. 4.3) to the account-rating-matrix **R** leads to inferior results because the reference recommender system suffers from all three shared account problems. We illustrate this with two toy examples. In both examples we consider the two users u_a and u_b that share the account s. User u_a has a known preference for the items a_1 and a_2 and user u_b has a known preference for the items b_1, b_2 and b_3. There are five candidate recommendations: $r_a^1, r_a^2, r_b^1, r_b^2$ and r_g. r_a^1 and r_a^2 are good recommendations for u_a. r_b^1 and r_b^2 are good recommendations for u_b. r_g is an overly general recommendation to which both users feel neutral.

Tables 4.1 and 4.2 summarize some intermediate computations on the first and second example respectively. The left hand side of both tables lists for every candidate recommendation (rows) the similarity to the known preferences of the shared account s (columns). The right hand side of both tables lists for every candidate recommendation (rows) three recommendation scores (columns). These scores are computed using Equation 4.1 and the similarity values on the left hand side of the respective row. The first two scores are for u_a and u_b respectively if they would not share an account. The third score is for s, the account shared by u_a and u_b.

The first example, corresponding to Table 4.1, illustrates that the item-based reference recommender system can suffer from the generality problem. From Table 4.1 we learn that if u_a would not share an account with u_b, the item-based reference recommender system would correctly assign the highest scores to r_a^1 and r_a^2 for u_a and to r_b^1 and r_b^2 for u_b. However, if u_a and u_b share the account s, the overly general item r_g receives the highest score. In this case, the item-based reference recommender system suffers from the generality problem because it does not discriminate between a recommendation score that is the sum of a few large contributions and a recommendation score that is the sum of many small contributions.

The second example, corresponding to Table 4.2, illustrates that the item-based reference recommender system can suffer from the dominance problem. From Table 4.2 we learn that if u_a would not share an account with u_b, the item-based reference recommender system would correctly assign the highest scores to r_a^1 and r_a^2 for u_a and to r_b^1 and r_b^2 for u_b. However, if u_a and u_b share the account s, all recommendations for u_b receive a higher score than any recommendation for u_a. Hence, the recommendations for u_b dominate the account at the expense of u_a. In this case, the item-based reference recommender system suffers from the

Table 4.1: Item similarities and resulting scores for Example 1.

	sim					s_{IB}		
	$(a_1,*)$	$(a_2,*)$	$(b_1,*)$	$(b_2,*)$	$(b_3,*)$	$(u_a,*)$	$(u_b,*)$	$(s,*)$
r_a^1	5	5	1	0	0	10	1	11
r_a^2	4	4	1	0	0	8	1	9
r_b^1	1	0	5	5	2	1	12	13
r_b^2	1	0	4	4	2	1	10	11
r_g	3	3	3	3	3	6	9	15

Table 4.2: Item similarities and resulting scores for Example 2.

	sim					s_{IB}		
	$(a_1,*)$	$(a_2,*)$	$(b_1,*)$	$(b_2,*)$	$(b_3,*)$	$(u_a,*)$	$(u_b,*)$	$(s,*)$
r_a^1	5	5	0	0	1	10	1	11
r_a^2	4	4	1	0	0	8	1	9
r_b^1	0	1	5	5	5	1	15	16
r_b^2	1	0	4	4	4	1	12	13
r_g	2	1	2	1	2	3	5	8

dominance problem because it does not take into account that u_b has more known preferences than u_a (3 vs. 2).

Both examples are suitable to illustrate that the reference recommender system suffers from the presentation problem. As an example, consider the first row of Table 4.1. The recommendation score $s(s, r_a^1) = 11$ is the sum of $sim(a_1, r_a^1) = 5$, $sim(a_2, r_a^1) = 5$ and $sim(b_1, r_a^1) = 1$. Therefore, it can be explained by a_1, a_2 and b_1. This is however a bad explanation because due to the presence of b_1, u_a will have difficulties to identify with the explanations and u_b might wrongly conclude that the recommendation is meant for her.

In our experimental evaluation (Sec. 4.9), we show that similar problems also arise for multiple large, real-life datasets.

4.5 Solving the Generality Problem

The previous section showed that the generality problem arises because the item-based reference recommender system (Eq. 4.1) does not discriminate between a score that is the sum of a few large similarities and a score that is the sum of many small similarities. Therefore, our first step is to adapt the item-based recommenda-

tion score (Eq. 4.1) into the *length-adjusted* item-based recommendation score:

$$\begin{aligned} s_{LIB}(u,i) &= s_{LIB}(I(u),i) \\ &= \frac{1}{|I(u)|^p} \cdot s_{IB}(I(u),i), \end{aligned} \tag{4.2}$$

with the hyperparameter $p \in [0,1]$. Although this adjustment does not immediately solve the generality problem, it does provide a way to differentiate between the sum of a few large similarities and the sum of many small similarities. By choosing $p > 0$, we create a bias in favor of the sum of a few large similarities. The larger p, the larger the bias. Also, notice that $|I(u)| = c(u)$.

Since the factor $\frac{1}{|I(u)|^p}$ is the same for all candidate recommendations i, the top N items for user u according to s_{LIB} and s_{IB} are the same. However, when we compare the scores of two different users, s_{LIB} also takes into account the the total amount of items preferred by the user.

To avoid the generality problem we ideally want to recommend an item i if it is highly relevant to one of the users in the userset of the shared account a. Hence, we want to compute the recommendation score of an item i for every individual user $u \in U(a)$, and use the highest one. Formally, we want to rank all items i according to their ideal recommendation score

$$\max_{u \in U(a)} s_{LIB}(I(u),i).$$

Unfortunately, we cannot compute this ideal recommendation score because $U(a)$ and consequently $I(u)$ are unknown. Instead, we only know $I(a) = \{j \in \mathcal{I} \mid \mathbf{R}_{aj} = 1\}$, the set of items preferred by account a.

We can, however, approximate the ideal recommendation score with its upper bound:

$$\max_{S \in 2^{I(a)}} s_{LIB}(S,i) \geq \max_{u \in U(a)} s_{LIB}(I(u),i),$$

in which $2^{I(a)}$ is the powerset of $I(a)$, i.e. the set containing all possible subsets of $I(a)$. The proposed approximation is an upper bound of the ideal score because every set of items $I(u)$ for which $u \in U(a)$ is also an element of $2^{I(a)}$. This approximation is based on the assumption that of all possible subsets of $I(a)$, the ones that correspond to users are more likely to result in the highest recommendation scores than the ones put together at random.

Consequently, we propose to solve the generality problem with the *disambiguating* item-based (DAMIB) recommender system, according to which the DAMIB recommendation score of an account a for an item i is given by:

$$s_{DAMIB}(a,i) = \max_{S \in 2^{I(a)}} s_{LIB}(S,i). \tag{4.3}$$

Every score $s_{DAMIB}(a,i)$ corresponds to an optimal subset $S_i^* \subseteq I(a)$:

$$S_i^* = \operatorname*{argmax}_{S \in 2^{I(a)}} s_{LIB}(S,i). \tag{4.4}$$

Hence, $s_{DAMIB}(a,i) = s_{LIB}(S_i^*, i)$. As such, the DAMIB recommender system not only computes the recommendation scores, but also finds the subset S_i^* that maximizes

the length-adjusted item-based recommendation score of a for i. This subset serves as the sharply defined, intuitive explanation for recommending i to a.

In other words, the DAMIB-recommender system implicitly splits the shared account a into (possibly overlapping) subsets S_i^* based on the intuitive and task-specific criterium that every S_i^* maximizes s_{LIB} for one of the candidate recommendations i. When $s_{LIB}(S_i^*, i)$ is high, we expect that S_i^* corresponds well to an individual user. When $s_{LIB}(S_i^*, i)$ is low, there is no user in the shared account for whom i is a strong recommendation and we expect S_i^* to be a random subset. As such, we avoid the error prone task of estimating the number of users in the shared account and explicitly splitting the account a into its alleged users, based on a general clustering criterium [118].

Furthermore, since subsets can potentially overlap, the DAMIB recommender system does not care whether the known preferences of the users in a shared account are strongly, slightly or not at all overlapping.

Finally, notice that for $p = 0$ it always holds that $s_{DAMIB} = s_{LIB} = s_{IB}$. Hence, the item based recommender system is a special case of the DAMIB recommender system.

4.6 Efficient Computation

Finding the maximum in Equation 4.3 in a direct way requires to compute s_{LIB} an exponential number of times, namely $2^{|I(a)|}$. Consequently, computing s_{DAMIB} in a direct way is intractable.

Fortunately, we are able to show a property of s_{LIB} that allows us to compute s_{DAMIB} in $\mathcal{O}\left(n \log n\right)$ time, with $n = |I(a)|$. This property is given by Theorem 4.6.1.

Theorem 4.6.1 *Let a be an account that prefers the set of items $I(a)$. Furthermore, let i be a candidate recommendation. If we rank all items $j, l \in I(a)$ such that $rank(j) < rank(l) \iff sim(j,i) > sim(l,i)$, then the subset $S_i^* \subseteq I(a)$ that maximizes $s_{LIB}(S,i)$ over all $S \in 2^{I(a)}$ is a prefix of that ranking.*

Proof: Given any $S \subseteq I(a)$. Initialize $P = S$. While P is not a prefix, remove r, the worst ranked item from P, and add a, the best ranked item that is not in P to P. As long as P is not yet a prefix, it holds that $sim(a,i) \geq sim(r,i)$. Therefore, every such item replacement increases (or keeps equal at least) $s_{LIB}(P,i)$ since the factor $1/|I(a)|^p$ does not change and a smaller term in the sum $\sum_{j \in I(a)} sim(j,i) \cdot |KNN(j) \cap \{i\}|$ is replaced by a larger term. Hence, for every $S \subseteq I(a)$ that is not a prefix of the ranking, we can always find a prefix $P \subseteq I(a)$ for which $s_{LIB}(P,i) \geq s_{LIB}(S,i)$. Therefore, the subset S_i^* that maximizes $s_{LIB}(S,i)$ over all $S \in 2^{I(a)}$ must always be a prefix of the ranking. □

Since the optimal subset is a prefix, we can find it with one scan over the ranked items of $I(a)$ in linear time. The logarithmic factor in the time complexity comes from ranking the $|I(a)|$ items.

This theorem is central to our approach because it allows us to compute s_{DAMIB} in $\mathcal{O}\left(n \log n\right)$ instead of exponential time.

4.7 Solving the Dominance Problem

The DAMIB recommender system allows us to detect when the dominance problem arises. This is because every recommendation i provided by DAMIB comes with a clear explanation in the form of the optimal subset $S_i^* \subseteq I(a)$. Therefore, if the union $\bigcup_{i \in top\text{-}N_a} S_i^*$ is only a small subset of $I(a)$, we know for sure that this small subset dominates the generation of the top N_a recommendations for account a.

Solving the dominance problem is done by choosing ALG = DAMIB in Algorithm 1, called COVER. As such, our final algorithm for recommending to shared accounts is DAMIB-COVER, with DAMIB-COVER(a) = COVER(a, DAMIB).

Algorithm 1: COVER(a,*ALG*)

input : $a \in \mathcal{A}$, ALG
output: top-N_a recommendations for account a

Compute $s_{ALG}(a,i)$ for all $i \in \mathcal{I} \setminus I(a)$
Rank all $i \in \mathcal{I} \setminus I(a)$ according to $s_{ALG}(a,i)$ in descending order with $t_a[r]$ the item at position r in the tuple of ranked items t_a
$C(a) \leftarrow \{\}$
$r \leftarrow 1$
$top\text{-}N_a \leftarrow \{\}$
while $|top\text{-}N_a| < N_a$ **do**
    $c \leftarrow t_a[r]$
    compute S_c^*
    if $D(S_c^*, C(a)) \geq \theta_D$ **then**
        $top\text{-}N_a \leftarrow top\text{-}N_a \cup \{c\}$
        $C(a) \leftarrow C(a) \cup S_c^*$
        remove c from t_a
        if $C(a) = I(a)$ **then**
            $C(a) \leftarrow \{\}$
            $r \leftarrow 1$
    else
        $r \leftarrow r + 1$
        if $r > |t_a|$ **then**
            $C(a) \leftarrow \{\}$
            $r \leftarrow 1$

The DAMIB-COVER algorithm uses the DAMIB scores to find the N_a highest scoring candidate recommendations and removes a candidate recommendation c from the top N_a if its explanation S_c^* is not sufficiently different from the explanations of the higher ranked candidates. The explanation-difference condition $D(S_c^*, C(a)) \geq \theta_D$ measures whether the explanation of a candidate (S_c^*) and the union of the explanations of the higher ranked candidates ($C(a)$) are sufficiently different.

Possible heuristic definitions of the explanation-difference condition are $|S_c^* \setminus C(a)| \geq 0.5 \cdot |S_c^*|$, and $|S_c^* \setminus C(a)| = |S_c^*|$. However, our experiments showed that

$|S_c^* \setminus C(a)| \geq 1$ works better than the other two. We therefore use the latter heuristic in the remainder of this work.

4.8 Solving the Presentation Problem

Generating the *top-N_a* recommendations for a shared account a with DAMIB-COVER is insufficient because the users that share the account don't know which recommendation belongs to which user. This is the presentation problem.

Our solution to the presentation problem is to present every recommendation $i \in$ *top-N_a* together with its explanation S_i^* as defined by Equation 4.4. We expect that for a large majority of the items i in the *top-N_a*, the explanation S_i^* is a subset of the preferences $I(u)$ of u, one of the user that shares the account a. We empirically validate this hypothesis in the experimental section (Sec. 4.9).

Hence, we can present the recommendations as *the item r is recommended to the person that prefers the items s_1, s_2 and s_3*. Then, a user will recognize s_1, s_2 and s_3 as her preferences, and know that r is recommended to her.

4.9 Experimental Evaluation

After introducing the datasets and competing algorithms, we discuss the performance of our novel algorithm.

4.9.1 Datasets

Ideally, we would use a dataset that contains real life shared account information. The CAMRa 2011 dataset, for example, contains household membership information for a subset of the users that rated movies [118]. As such we could construct realistic shared accounts with this dataset. Unfortunately, the owner did not wish to distribute the dataset anymore and we have no knowledge of other datasets that contain shared account information. However, from the CAMRa 2011 dataset we learn that most household accounts consist of two users (272 out of 290 households) and some consist of three (14 out of 290) or four users (4 out of 290). Therefore, we will follow the approach of Zhang et al. and create 'synthetic' shared accounts by randomly grouping users in groups of two, three or four [118]. Although this approach is not perfect, Zhang et al. showed that the properties of the 'synthetic' shared accounts were similar to the properties of the real shared accounts from the CAMRa 2011 dataset [118].

We evaluated our proposed solution on four datasets: the *Yahoo!Music* [113], *Movielens1M* [32], *Book-Crossing* [122] and the *Wiki10+* [123] datasets.

The ***Yahoo!Music*** dataset contains ratings of 14382 users on 1000 songs on a 1 to 5 scale [113]. Since we consider the problem setting with binary, positive-only data we binarize the ratings. We convert the ratings 4 and 5 to preferences and ignore all other ratings. On average, a user has 8.7 preferences.

The ***Movielens1M*** dataset contains ratings of 6038 users on 3533 movies on a 1 to 5 scale [32]. Again, we convert the ratings 4 and 5 to preferences and ignore all other ratings. On average, a user has 95.3 preferences.

The ***Book-Crossing*** dataset contains two sorts of information [122]. First, there are ratings of users for books on a 1 to 10 scale. Analogously to the previous two datasets, we convert the ratings 8,9 and 10 to preferences and ignore all other ratings. Secondly, there are also binary preferences that we simply add to our list of preferences. In total, there are 87 835 users, 300695 books and every user has on average 11 preferences.

The ***Wiki10+*** dataset contains 99162 tags assigned to 20 751 Wikipedia articles [123]. In this case we consider the recommendation of tags to articles, hence the articles take the role of 'users' and the tags take the role of 'items'. If an article a was tagged at least once with a tag t, we consider a 'preference' of article a for tag t. In this context, a shared account is a big article on a wider topic containing multiple smaller 'articles' on subtopics. On average, every article has 22.1 tags.

4.9.2 Competing Algorithms

We compare our novel algorithm, DAMIB-COVER, with two competing algorithms. The first one is IB, simply the item-based reference recommender system applied to the account-rating-matrix, essentially ignoring the existence of the shared account problems. This is our baseline. The second competing algorithm is IB-COVER, which is defined as IB-COVER(a) = COVER(a,IB). IB-COVER is similar to one of the algorithms already proposed by Yu et al. in a different context [117].

4.9.3 Performance

First, consider the recall of a user that shares an account a with $|U(a)|$ other user. This is the percentage of its individual top-5 recommendations that is also present in the top-N_a recommendations for its shared account, with $N_a = 5 \cdot |U(a)|$. Formally, we define the recall of user u as:

$$rec(u) = \frac{|top\text{-}5(u) \cap top\text{-}N_a(a)|}{5}.$$

Ideally, the recall of all users in a shared account is 1, meaning that the top-N_a for the shared account is the union of the individual top-5's of the $|U(a)|$ users sharing the account.

Now, to investigate how many users genuinely suffer from sharing an account, we measure the fraction of users that does not get any relevant recommendation, i.e. that does not find a single one of its top-5 individual recommendations in the top-N_a recommendations of the shared account it belongs to. We denote this number as $rec_0^{\mathcal{U}}$, the fraction of users for which the recall is zero. Formally, we define

$$rec_0^{\mathcal{U}} = \frac{|\{u \in \mathcal{U} \mid rec(u) = 0\}|}{|\mathcal{U}|}.$$

An illustrative example of a user that genuinely suffers from sharing an account is depicted in Table 4.3. This table shows two real users from the *Movielens1M* dataset with their respective known preferences $I(u)$ and item-based individual top-5 recommendations. Their item-based individual top-5 recommendations look reasonable given their known preferences and it is not unrealistic that these two users would be part of the same household and therefore share an account.

Table 4.3: Example of user 562 suffering from sharing an account with user 4385.

user ID	562	4385
$I(u)$	Wes Craven's New Nightmare, The Exorcist III, Serial Mom, Scream, Scream 2, The Blair Witch Project, Good Will Hunting, Misery, Interview with the Vampire, Candyman, Freddy's Dead: The Final Nightmare	American Beauty, The Shawshank Redemption, Being John Malkovich, L.A. Confidential, Boys Don't Cry, Croupier, Dogma, Cider House Rules, Girl Interrupted, Saving Grace, The Talented Mr. Ripley
individual top-5: IB, $k = 25$	A Nightmare on Elm Street, Halloween, Halloween:H20, The Shining, Seven	Pulp Fiction, Fargo, The Sixth Sense, The Silence of the Lambs, Shindler's List
R_{sa} = IB	The Silence of the Lambs, Fargo, Pulp Fiction, The Sixth Sense, Saving Private Ryan, The Usual Suspects, Shindler's List, Shakespeare in Love, Star Wars: Episode V, The Matrix	
R_{sa} = DAMIB-COVER (p=0.75)	The Silence of the Lambs, Fargo, Schindler's List, A Nightmare on Elm Street, Halloween:H20, Pulp Fiction, Shakespeare in Love, The Shining, The Exorcist, Sleepy Hollow	

Consequently, Table 4.3 also shows the recommendations for the 'synthetic' account shared by both users for two cases: R_{sa} = IB and R_{sa} = DAMIB-COVER. In case R_{sa} = IB, $rec(562) = 0$, i.e. user 562 does not get a single recommendation and genuinely suffers from sharing an account. In case R_{sa} = DAMIB-COVER, $rec(562) = 0.6$, i.e. user 562 gets 3 good recommendation and there is no serious problem. Obviously, this is just one example and we need to look at all users in the dataset for comparing the different algorithms.

Figure 4.1 displays $rec_0^{\mathcal{U}}$ for the *Yahoo!Music* dataset. The number of nearest neighbors, k, is a parameter of the item-based reference recommender system (Eq 4.1). There are multiple ways of choosing k. Amongst others, examples are accuracy in an off-line experiment, subjective quality judgment of the recommendations, accuracy in an on-line A/B test, computational efficiency, etc. Therefore, we present our results for a variation of reference recommender systems, i.e. item-based collaborative filtering recommender systems that use cosine similarity and differ in their choice of k. Consequently, every plot in Figure 4.1 shows the results for a different k.

For every choice of k and the individual top-5 recommendations corresponding to this choice we consider four experiments: an account shared by one, two, three or four users respectively. Notice that an account shared by one user is actually not shared. Every horizontal axis indicates the number of users that share the account, every vertical axis indicates the resulting $rec_0^{\mathcal{U}}$. The four different markers show the results for four different shared account recommender systems R_{sa}: the baseline algorithm IB, the competitor IB-COVER and two variations of the proposed DAMIB-COVER algorithm. These two variations differ in their choice of the hyperparameter

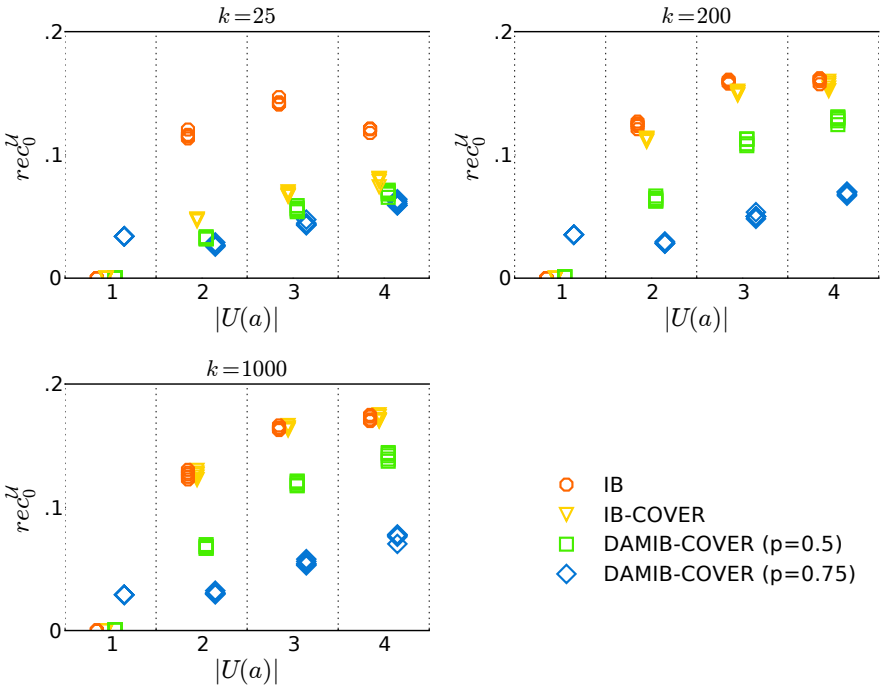

Figure 4.1: $rec_0^{\mathcal{U}}$ as a function of the number of merged users, $|U(a)|$, for different k and shared account recommender systems R_{sa} on the *Yahoo!Music* dataset. Lower is better. 95% confidence intervals are more narrow than the marker clouds and are therefore not drawn.

p (Eq. 4.2): $p = 0.5$ and $p = 0.75$. Since we repeat every experiment 5 times with other randomizations, every plot contains $5 \times 4 = 20$ markers of the same kind. However, because of the low spread, most markers are plotted on top of each other, forming dense marker clouds. Furthermore, since the 95% confidence intervals for the mean are more narrow than the marker-clouds of 5 data-points, we do not draw them. Consequently, two marker clouds that are visually well separated, are also significantly different at the 5% significance level.

We make four observations from Figure 4.1. First, we observe that the baseline performance is not good. Up to 19% of the users in this dataset get no relevant recommendation when they share their account with another user. This confirms that shared accounts can cause significant problems for recommender systems.

Secondly, our proposed solution, the DAMIB-COVER algorithm, can significantly improve $rec_0^{\mathcal{U}}$. In some cases the improvement is even drastic. One example is for the case that $|U(a)| = 2$ and that the individual top-5 is generated with $k = 200$. In this case, 12% of the users does not get any relevant recommendation when using the baseline algorithm IB. By using DAMIB-COVER ($p = 0.75$), this number is reduced with a factor four ($rec_0^{\mathcal{U}} = 0.03$).

Thirdly, sometimes IB-COVER already improves over IB. There are however multiple cases in which DAMIB-COVER further improves over IB-COVER. Furthermore,

the advantages of DAMIB-COVER over IB-COVER will become even more evident in the evaluation of the presentation problem in Section 4.9.5.

Finally, when $|U(a)| = 1$, i.e. when the accounts are not shared, $rec_0^{\mathcal{U}} = 0$ by definition for the baseline algorithm IB. However, we observe that also for the IB-COVER and the variants of the DAMIB algorithms $rec_0^{\mathcal{U}}$ can be kept sufficiently low. Hence, the proposed DAMIB algorithm does not fail when accounts are not shared.

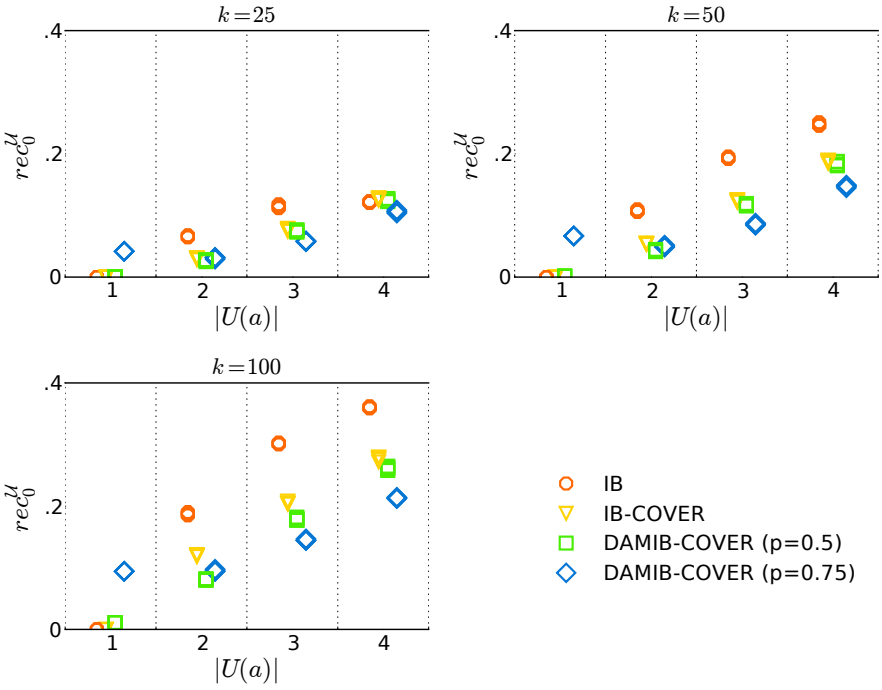

Figure 4.2: $rec_0^{\mathcal{U}}$ as a function of the number of merged users, $|U(a)|$, for different k and shared account recommender systems R_{sa} on the *Wiki10+* dataset. Lower is better. 95% confidence intervals are more narrow than the marker clouds and are therefore not drawn.

Similarly, figures 4.2-4.4 display $rec_0^{\mathcal{U}}$ for the *Wiki10+*, *Book-Crossing* and *Movielens1M* datasets respectively. From these figures we learn that other datasets display similar trends. Furthermore, we observe a worst case performance as bad as 60%, i.e. 60% of the users does not get any relevant recommendation when they share their account with other users (Fig. 4.3). Finally, Figure 4.4 shows that our algorithm does not always improve $rec_0^{\mathcal{U}}$.

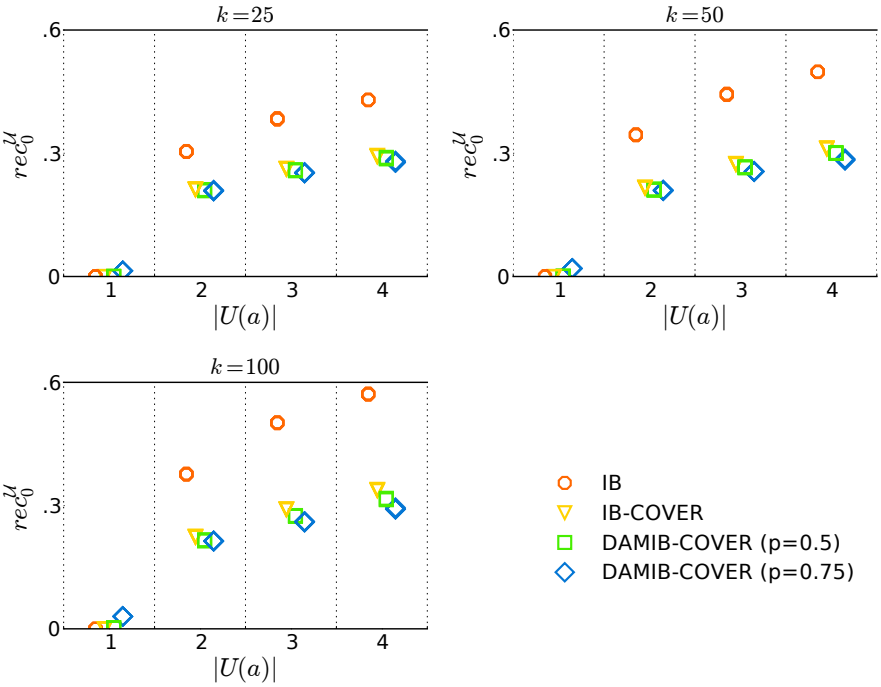

Figure 4.3: $rec_0^{\mathcal{U}}$ as a function of the number of merged users, $|U(a)|$, for different k and shared account recommender systems R_{sa} on the *Book-Crossing* dataset. Lower is better. 95% confidence intervals are more narrow than the marker clouds and are therefore not drawn.

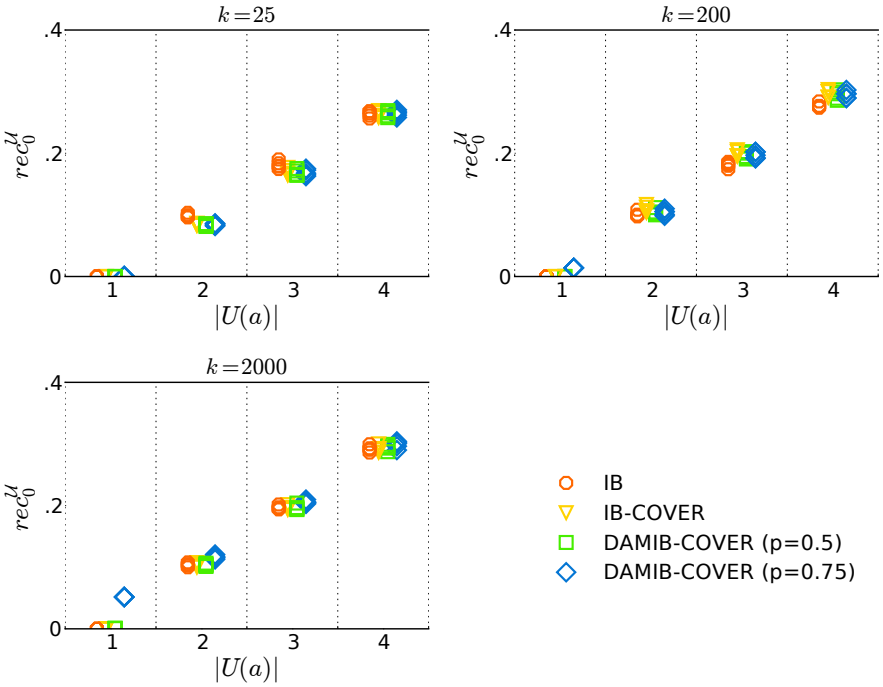

Figure 4.4: $rec_0^{\mathcal{U}}$ as a function of the number of merged users, $|U(a)|$, for different k and shared account recommender systems R_{sa} on the *Movielens1M* dataset. Lower is better. 95% confidence intervals are more narrow than the marker clouds and are therefore not drawn.

4.9.4 Limited Trade-Off

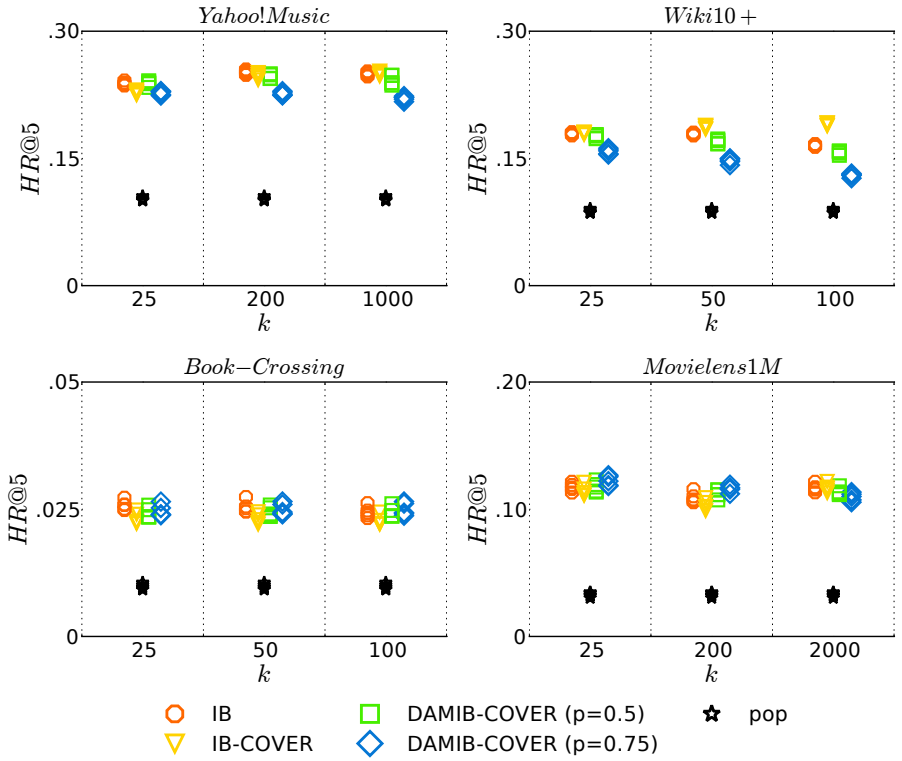

Figure 4.5: HR@5 as a function of k for different recommender systems. Higher is better.

To emphasize the point that the DAMIB-COVER algorithm still performs well in a traditional setting when no accounts are shared, we also discuss the results of DAMIB-COVER on a more established experimental setup that was used by Deshpande et al. [19], amongst many others. To avoid all confusion: this experimental setup has nothing to do with shared accounts. In this experimental setup, one preference of every user is randomly chosen to be the test preference h_u for that user. If a user has only one preference, no test preference is chosen. The remaining preferences are represented as a 1 in the training matrix $\mathbf{R}$ (which is in this case exactly the same as $\mathbf{T}$ because no accounts are shared). All other entries of $\mathbf{R}$ are zero. We define $\mathcal{U}_t$ as the set of users with a test preference. For every user $u \in \mathcal{U}_t$, every algorithm ranks the items $\{i \in \mathcal{I} \mid \mathbf{R}_{ui} = 0\}$ based on $\mathbf{R}$. Following Deshpande et al. we evaluate every ranking using hit rate at 5 [19]. Hit rate at 5 is given by

$$HR@5 = \frac{1}{|\mathcal{U}_t|} \sum_{u \in \mathcal{U}_t} |\{h_u\} \cap top5(u)|,$$

with $top5(u)$ the 5 highest ranked items for user u. Hence $HR@5$ gives the percentage of test users for which the test preference is in the top 5 recommendations. The

results of the experiment for all datasets are shown in Figure 4.5. Additionally to the algorithms discussed earlier, Figure 4.5 also contains the results for the baseline-algorithm POP, the non-personalized algorithm that ranks all items according to their popularity, i.e. the number of users in the training set that prefer the item. Also in this case we repeated every experiment five times with a different randomization. Again, the five data points are often plotted on top of each other because of the low spread. Figure 4.5 shows that $HR@5$ is very similar for DAMIB-COVER and IB. Hence, there is almost no trade-off in terms of global accuracy measured as $HR@5$.

4.9.5 Presentation

In Section 4.8 we proposed to solve the presentation problem by presenting every recommendation together with its explanation. If then, a user in the shared account recognizes an explanation as a subset of her preferences, this user can identify with the recommendation and therefore knows the recommendation is meant for her. For this proposed solution to work, it is crucial that the recommendation is identifiable, i.e. that its explanation is a subset of the preferences of one of the users in the shared account. We quantify the identifiability of a recommendation i, with explanation S_i^*, for a shared account a as:

$$ident(S_i^*) = \max_{u \in U(a)} \frac{|S_i^* \cap I(u)|}{|S_i^*|}.$$

Ideally, $ident(S_i^*) = 1$, i.e. every item in the explanation is a preference of one and the same user. In the worst case, $ident(S_i^*) = 1/|U(a)|$, i.e. the explanation contains an equal amount of preferences from all users in the shared account and therefore none of the users can identify herself with the recommendation.

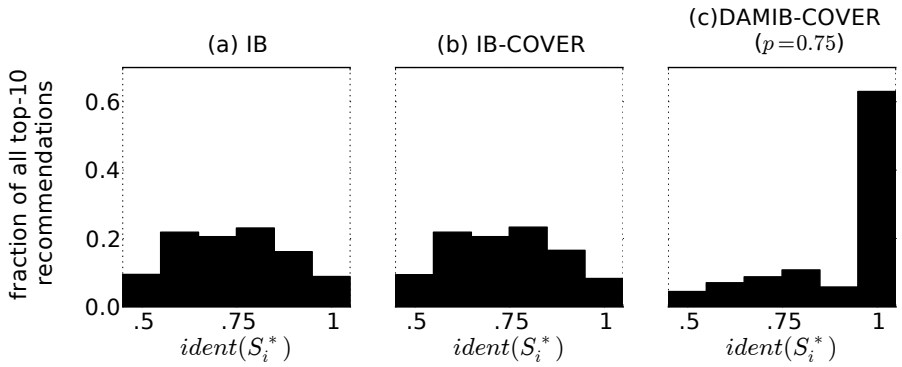

Figure 4.6: Histograms of identifiability of top-10 recommendations for $|U(a)| = 2$ on the *Yahoo!Music* dataset. $k = 200$.

Figure 4.6 shows histograms of the identifiability of the top-10 recommendations for $|U(a)| = 2$ on the *Yahoo!Music* dataset for multiple shared account recommender systems R_{sa}. From Figure 4.6a we learn that if one simply applies the item-based reference algorithm to the shared account data of the *Yahoo!Music* dataset, the presentation problem arises: very few recommendations can be identified with one

of the users in the shared account, i.e. $ident(S_i^*) = 1$ for only 10% of the explanations. Figure 4.6b shows that using IB-COVER instead of IB does not improve the situation. However, figure 4.6c shows that using DAMIB-COVER drastically increases the identifiability of the recommendations, i.e. $ident(S_i^*) = 1$ for approximately 60% of the explanations. Hence, the DAMIB explanations are superior to the item-based explanations.

Figures 4.7-4.9 show that the other datasets display the same trend, although less pronounced.

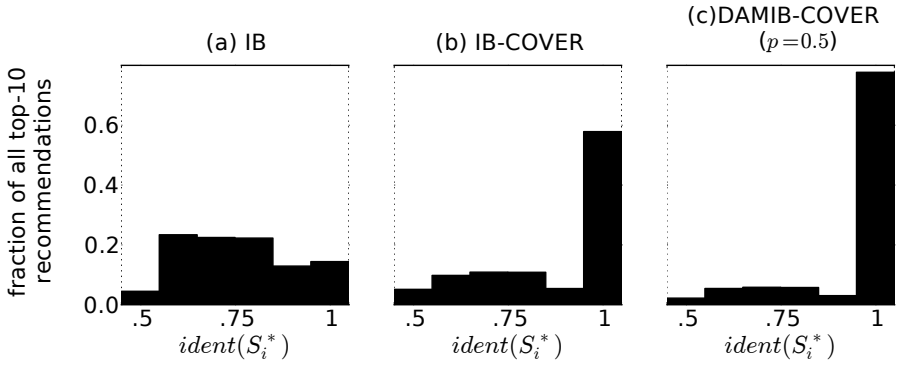

Figure 4.7: Histograms of identifiability of top-10 recommendations for $|U(a)| = 2$ on the *Wiki10+* dataset. $k = 100$.

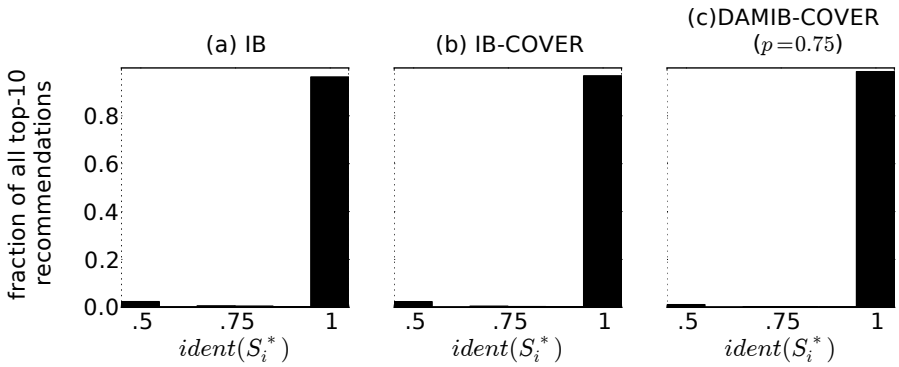

Figure 4.8: Histograms of identifiability of top-10 recommendations for $|U(a)| = 2$ on the *Book-Crossing* dataset. $k = 100$.

4.10 Related Work

Although we did not find prior art tackling the same challenges as we do, there are some works that have commonalities with ours.

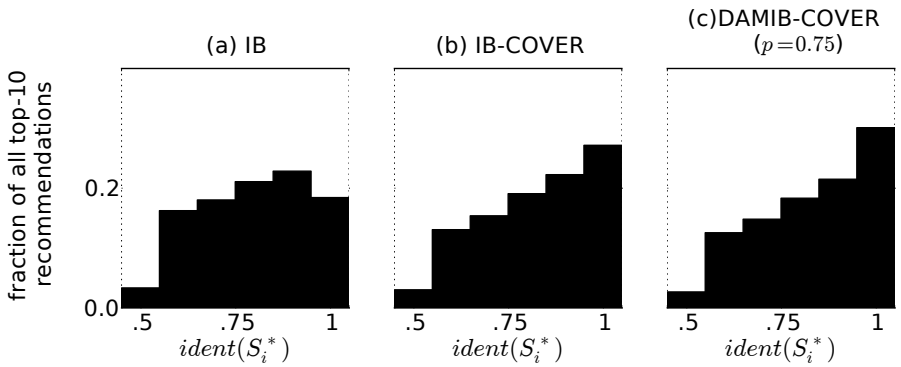

Figure 4.9: Histograms of identifiability of top-10 recommendations for $|U(a)| = 2$ on the *Movielens1M* dataset. $k = 25$.

First, Palmisano et al. [63] consider a problem setting in which the contextual information is *sometimes* missing. However, their proposed solution draws upon all training data for which they do know the context to devise a 'context predictor'. Hence, their solution relies on contextual information.

Second, Anand et al. [3] do not use explicit contextual information. However, their solution assumes that the preferences of every account are grouped into transactions. Our solution does not assume that this kind of extra data is available.

Third, Zhang et al. [118] study the extent to which it is possible to explicitly split shared accounts into their individual users without contextual information. They are able to split certain shared accounts very nicely, but find that in general, explicitly splitting accounts into their users is very error prone. Fortunately, by means of s_{DAMIB}, we are able to avoid this explicit split of accounts into users and perform a softer, implicit split instead.

Fourth, Yu et al. [117] propose to use the explanations of item-based recommendations to generate diversified top-N recommendation lists. Where they focus on the diversity of the explanations, we focus on covering all items preferred by the account with the different explanations. Furthermore, in our experimental evaluation (Sec. 4.9) we showed that the explanations provided by our DAMIB-COVER algorithm are superior to those that can be extracted from an item-based algorithm.

Finally, as mentioned in Section 2.2.6, the maxMF [109] and NLMF [45] methods might be an alternative to solve the generality problem. However, these methods do not solve the dominance an presentation problems.

4.11 Conclusions

We showed that the widely used item-based recommender system fails when it makes recommendations for shared accounts. Therefore, we introduced the challenge of Top-N recommendation for shared accounts in the absence of contextual information. Furthermore, we proposed the DAMIB-COVER algorithm, our solution to this challenge. Central to our approach, we showed a theorem that allowed us to compute a recommendation score in $\mathcal{O}(n \log n)$ instead of exponential time. Finally,

we experimentally validated that our proposed solution has important advantages. An important direction for future work, is to generalize our proposed solution to a wider range of reference recommender systems.

CHAPTER 5

Efficiently Computing the Exact k-NN Graph for High Dimensional Sparse Data

Neighborhood-based collaborative filtering methods with an analytically solvable deviation function require the computation the k most similar users(items) to every user(item) in the dataset. Efficiently computing these neighborhoods is an application of the more general problem of efficiently computing the exact k-nearest neighbors graph for high dimensional sparse data.

In this chapter[1]*, we are given a large collection of sparse vector data in a high dimensional space; and we investigate the problem of efficiently computing the k most similar vectors to every vector. Two common names for this task are computing the k-nearest neighbors graph and performing a k-nearest neighbors self-join. Currently, exact, efficient algorithms exist for low dimensional data. For high dimensional data, on the other hand, only approximate, efficient algorithms have been proposed. However, the existing approaches do not discriminate between dense and sparse data. By focusing on sparse data, we are able to propose two inverted index-based algorithms that exploit the sparseness in the data to more efficiently compute the exact k-nearest neighbors graph for high dimensional data. We first propose a simple algorithm based on dynamic indexing. Afterwards, we propose a second algorithm that extends the first one by introducing a virtual threshold that enables the exploitation of various upper bounds for pruning candidate neighbors. We experimentally show that our approaches result in significant speedups over state-of-the-art approaches.*

[1] This chapter is based on work under submission as "Efficiently Computing the Exact k-NN Graph for High Dimensional Sparse Data" by Koen Verstrepen and Bart Goethals [103].

5.1 Introduction

Computing the k most similar vectors to every vector in a large collection of sparse, high dimensional vectors, is a crucial component of many real world problems. On the one hand, objects in the real world are often represented as high dimensional vectors. Documents, paragraphs, tweets and search snippets, for example, are typically represented by a vector, where each dimension of the vector represents a word in a large vocabulary, and the value in each dimension indicates the importance of the word in the document. These document vectors are very sparse, since every document only contains a small subset of the vocabulary. Other objects that are typically represented by high dimensional vectors are users that buy products, listen to songs, watch movies, etc. In this case, each dimension of the vector corresponds to a product, song, movie, etc. in a large collection, and the value in each dimension is the number of times that the user interacted with the item corresponding to the dimension. Also these vectors are very sparse, since even popular objects are typically only interacted with by a small fraction of all users.

On the other hand, many problems require the computation of the k-nearest neighbors graph. Examples are:

- When **exploring** a collection of objects, the k-nearest neighbors graph can be queried to find the objects most similar to the currently considered object.
- The k-nearest neighbors graph of existing objects can be used to **rapidly search for the nearest neighbors of novel objects** [33].
- Many **clustering** algorithms take the similarity (or distance) matrix of the objects as input [34].
- The k-nearest neighbors graph can be used for link prediction in (social) networks [52].
- Item-based (User-based) **collaborative filtering algorithms** use the k-nearest neighbors graph of items (users) [20].

A naïve baseline algorithm for computing the k-nearest neighbors graph, which is another name for performing a k-nearest neighbors self-join, computes all n^2 pairwise object similarities, where n is the number of objects in the collection. This is intractable for many real world problems given the large size of their corresponding object collections.

For low dimensional data, more efficient solutions have been proposed [69, 16]. For high dimensional data, on the other hand, only approximate algorithms have been proposed [14, 23, 70]. Although sometimes approximate results are sufficient, often exact results are required. For example, when a user searches for papers similar to a target paper, she will not be satisfied if the approximate results do not contain this one very similar paper she already knows.

Now, these approaches do not discriminate between dense and sparse data, although sparseness, if present, could be exploited to find an exact instead of an approximate solution.

Therefore, we propose two novel, inverted index-based algorithms that are able to exploit the sparseness in the data to efficiently compute the k-nearest neighbors

graph without relying on approximate techniques. Our first proposal is a simple algorithm based on the dynamic indexing technique proposed by Sarawagi and Kirpal [78]. Our second algorithm extends the first one by introducing a virtual threshold that enables the exploitation of various upperbounds for pruning candidate neighbors. These upperbounds were proposed for solving the related problem called *all-pairs similarity search* or *self-similarity join*, i.e. finding all pairs of objects more similar than a certain minimal similarity [4, 7].

Although there will still exist large datasets for which only an approximate solution is within reach, our proposed algorithms increase the maximum size of sparse datasets for which it is possible to compute the exact k-NN graph in a reasonable amount of time. Finally, we experimentally validate that both algorithms realize speedups over to the current state-of-the-art.

5.2 Problem Statement

Given a set of high-dimensional, sparse, positive vectors $\mathcal{V} = \{\mathbf{v} \mid \mathbf{v} \in \mathbb{R}^m_{\geq 0}\}$, we solve the problem of finding for every vector $\mathbf{v} \in \mathcal{V}$ the k most similar vectors $\mathbf{w} \in \mathcal{V}$ according to the similarity function $sim(\mathbf{v}, \mathbf{w})$.

To cover the cases in which the k-th most similar vector is equally similar as the $(k+x)$-th most similar vector, we more formally define finding the top-k of every $\mathbf{v} \in \mathcal{V}$ as finding for every $\mathbf{v} \in \mathcal{V}$ all vectors $\mathbf{w} \in \mathcal{V} \setminus \{\mathbf{v}\}$ for which it holds that $|\{\mathbf{u} \in \mathcal{V} \setminus \{\mathbf{v}, \mathbf{w}\} \mid sim(\mathbf{v}, \mathbf{u}) > sim(\mathbf{v}, \mathbf{w})\}| < k$, i.e. there are less than k vectors that are more similar to $\mathbf{v}$ than $\mathbf{w}$. In the remainder of this chapter, we will call this result the top-k, although it can contain more than k vectors in this special case. Also, we define $kNN(\mathbf{v})$ as the data structure containing the pairs $(\mathbf{w}, sim(\mathbf{v}, \mathbf{w}))$ for every vector $\mathbf{w}$ that is in the top-k of the vector $\mathbf{v}$.

Furthermore, we focus on cosine similarity, which is often used on high dimensional, sparse vectors. For $\mathbf{v}, \mathbf{w} \in \mathcal{V}$, it is defined as

$$\cos(\mathbf{v}, \mathbf{w}) = \frac{\mathbf{v} \cdot \mathbf{w}}{||\mathbf{v}||_2 \cdot ||\mathbf{w}||_2},$$

which can be rewritten as a dot product of vectors that are normalized to unit length in a preprocessing step:

$$\cos(\mathbf{v}, \mathbf{w}) = \frac{\mathbf{v}}{||\mathbf{v}||_2} \cdot \frac{\mathbf{w}}{||\mathbf{w}||_2}.$$

In the remainder of this chapter, we will assume that all vectors in $\mathcal{V}$ are unit length normalized.

Since we consider sparse, high dimensional vectors, the value of a vector $\mathbf{v} \in \mathcal{V}$ in dimension i, denoted $\mathbf{v}[i]$, is mostly zero. Therefore, we use sparse vector representations in our implementations. The sparse representation of a vector $\mathbf{v}$ is the set of pairs $(i, \mathbf{v}[i])$ for which the integer $i = 1 \ldots m$ and $\mathbf{v}[i] > 0$. We define the size of a vector $\mathbf{v}$, denoted $|\mathbf{v}|$, as the number of pairs in this set. Similarly, we define the size of a dimension i, denoted $c(i)$, as the number of pairs in the set containing all pairs $(i, \mathbf{v}[i])$ for which $\mathbf{v} \in \mathcal{V}$ and $\mathbf{v}[i] > 0$. The vector size $|\mathbf{v}|$ should not be confused with the vector norm $||\mathbf{v}||_2$.

Finally, note that $\mathbf{v}$(bold) is a vector and that v is the identifier of this vector.

Table 5.1: Basic properties of the datasets.

	dblp	*tweets*	*msd*
$\lvert\mathcal{V}\rvert$	629 814	1 000 010	1 019 318
m	740 973	630 784	384 546
$\sum_{\mathbf{v}\in\mathcal{V}} \lvert\mathbf{v}\rvert$	6 507 829	8 491 045	48 373 586
$\max_{i=1\ldots m}\left(\frac{c(i)}{\lvert\mathcal{V}\rvert}\right)$	0.32	0.44	0.11

5.3 Real World Datasets

In this chapter, we use three real-world datasets: *dblp* , *tweets* and *msd* . Table 5.1 summarizes some basic properties of the datasets.

The *dblp* dataset [94] contains 629 814 scientific papers that are represented by their title and authors. Hence, every paper is a sparse, high dimensional vector in which every dimension corresponds to a word in the vocabulary of 740 973 words, including author names. If the word that corresponds to dimension i appears in the title of paper $\mathbf{v}$ or is the name of one of the authors, $\mathbf{v}[i] = 1$. Otherwise, $\mathbf{v}[i] = 0$. Similar datasets were constructed by Arasu et al. [5], Bayardo et al. [7], and Dong et al. [23].

We constructed the *tweets* dataset with the Twitter streaming API [98]. To gather a sample of random English tweets, we selected tweets that contained at least one out of 70 English stop words with a minimal length of 4 characters, from the Natural Language Toolkit [61]. From the selected tweets, we removed the stop words and stemmed the remaining words. As such, we obtained a dataset with 1 000 010 tweets in which every tweet is represented by a sparse, high dimensional vector in which every dimension corresponds to a word in the vocabulary of 630 784 words. If the word that corresponds to dimension i appears in tweet $\mathbf{v}$, $\mathbf{v}[i] = 1$. Otherwise, $\mathbf{v}[i] = 0$.

The *msd* dataset is derived from the *Echo Nest Taste Profile Subset* of the *Million Song Data Challenge* [10] and contains 1 019 318 people that are represented by the songs they listened to. Hence, every person is a sparse, high dimensional vector in which every dimension corresponds to a song in the catalog of 384 546 songs. If person $\mathbf{v}$ listened c times to the song that corresponds to dimension i, $\mathbf{v}[i] = c$. If $\mathbf{v}$ did not listen to the song, $\mathbf{v}[i] = 0$.

5.4 Naïve Baseline

The naïve approach to computing the k-nearest neighbors graph is layed out in Algorithm 2. This algorithm computes for every vector $\mathbf{v}$ the similarity with every other vector $\mathbf{w}$ by explicitly computing the dotproduct $\mathbf{v}\cdot\mathbf{w}$ (Line 4). Since the dotproduct is between sparse vectors, it can be efficiently computed by means of the well known method layed out in Algorithm 3. Afterwards, it updates $kNN(\mathbf{v})$ accordingly (Line 5, Alg. 2). Hence, the naïve baseline algorithm explicitly computes $|\mathcal{V}|^2 - |\mathcal{V}|$ (sparse) dotproducts, which is intractable for many real world datasets.

Algorithm 2: Naïve Baseline

```
   input  : 𝒱, k
   output : For every v ∈ 𝒱 : kNN(v)
1  for each v ∈ 𝒱 do
2      s ← 0_{|𝒱|×1}
3      for each w ∈ 𝒱 \ {v} do
4          s[w] ← v · w
5      kNN(v) ← ExtractNN(s, k)
```

Algorithm 3: Sparse Dotproduct

```
   input  : v, w
   output : v · w
1  dotp ← 0
2  i ← smallest i  s.t. v[i] > 0
3  j ← smallest j  s.t. w[j] > 0
4  while i < m and j < m do
5      if i < j then
6          i ← next i  s.t. v[i] > 0 or m
7      else if i > j then
8          j ← next j  s.t. w[j] > 0 or m
9      else
10         dotp ← dotp + v[i] · w[j]
11         i ← next i  s.t. v[i] > 0 or m
12         j ← next j  s.t. w[j] > 0 or m
13 v · w ← dotp
```

5.5 Basic Inverted Index

Inspired by the top-k document retrieval algorithms [97], we propose to speed-up the exact computation of the k-nearest neighbors graph by exploiting the sparseness in the data. The sparseness can be exploited by means of an inverted index representation of the dataset [111]. In our case, the inverted index representation of $\mathcal{V}$ is $\mathcal{L}$, a list of m inverted lists $L_1, L_2, ..., L_m$, where each inverted list corresponds to one of the m dimensions and list L_i contains all pairs $(v, \mathbf{v}[i])$ for which $\mathbf{v} \in \mathcal{V}$, v is the identifier of $\mathbf{v}$, and $\mathbf{v}[i] > 0$. Algorithm 4 describes a basic inverted index-based approach.

This approach is more efficient because it avoids the wasteful explicit computation of the dotproduct on Line 4 of the naïve baseline algorithm (Alg. 2). Instead, this algorithm loops over the $|\mathbf{v}|$ non-zero dimensions in the sparse representation of $\mathbf{v}$ (Line 5) and for each of these dimensions it loops over the inverted list corresponding to them. As such, this algorithm avoids testing whether a dimension is non-zero in both $\mathbf{v}$ and $\mathbf{w}$, as is done in Algorithm 3. To illustrate this, consider the two vectors in Table 5.2 that belong to a large collection of vectors $\mathcal{V}$. Computing the

Algorithm 4: Basic Inverted Index-based algorithm

```
   input  : V, k
   output : For every v ∈ V : kNN(v)
1  Build the inverted index L
2  for each v ∈ V do
3      s ← 0_{|V|×1}
4      U ← {}
5      for each i = 1...m s.t. v[i] > 0 do
6          for each (w, w[i]) ∈ L_i do
7              s[w] ← s[w] + v[i] · w[i]
8              U ← U ∪ {w}
9      kNN(v) ← ExtractNN(s, U, k)
```

Table 5.2: Sparse representation of two vectors selected from a large dataset $\mathcal{V}$.

$\mathbf{x} = \{(1, 0.58), (2000, 0.58), (4000, 0.58)\}$
$\mathbf{y} = \{(1000, 0.5), (3000, 0.5), (5000, 0.5), (30000, 0.5)\}$

k-nearest neighbors of $\mathbf{x}$ with the naïve baseline algorithm, requires one to compute the dotproduct between $\mathbf{x}$ and $\mathbf{y}$ at some point. This is typically done by Algorithm 3, which initializes i to 1 and j to 1000, corresponding to the pairs $(1, 0.58)$ and $(1000, 0.5)$ respectively. Then, the dimensions 1 and 1000 are compared and found to be different. Therefore, i is incremented to point to $(2000, 0.58)$, which is the next non zero entry of $\mathbf{x}$. Then, again, the dimensions 2000 and 100 are compared and found to be different. Therefore, j is incrmented to now point to $(3000, 0.5)$. This continues until i reaches the end of the sparse representation of $\mathbf{x}$. Hence, a lot of value comparisons and iterator increments are required to find out that the dotproduct between $\mathbf{x}$ and $\mathbf{y}$ is zero. The Basic Inverted Index-based algorithm, on the other hand, does not even notice that $\mathbf{y}$ exists when it computes the k-nearest neighbors of $\mathbf{x}$ because $\mathbf{y}$ is not present in L_1, L_{2000} or L_{4000}. Consequently, it does not waste any computations on it.

Furthermore, this inverted index-based algorithm maintains a list U of all vectors that have a non-zero similarity with v to speed-up the top-k extraction on Line 9.

5.6 DynamicIndex

Instead of building the inverted index in a preprocessing step, as done by the basic inverted index-based algorithm (Line 1) and as typically done by top-k document retrieval algorithms [97], we adopt the idea of Sarawagi and Kyrpal to build it dynamically [78]. This allows us to propose the *DynamicIndex* algorithm (Algorithm 5).

Building the inverted index $\mathcal{L}$ dynamically, means that it is initialized with m empty inverted lists (Line 1) and that a new vector is added only after it has been processed (Line 13 and 14). The advantage of building the inverted index dynamically

Algorithm 5: DynamicIndex

input : $\mathcal{V}$, k
output : For every $\mathbf{v} \in \mathcal{V}$: $kNN(\mathbf{v})$, $\mathcal{L}$

```
1  ∀L_i ∈ 𝓛 : L_i ← {}
2  ∀v ∈ 𝒱 : kNN(v) ← empty heap
3  for each v ∈ 𝒱 do
4      s ← 0_{|𝒱|×1}
5      U ← {}
6      for each i = 1...m s.t. v[i] > 0 do
7          for each (w, w[i]) ∈ L_i do
8              s[w] ← s[w] + v[i] · w[i]
9              U ← U ∪ {w}
10     kNN(v) ← ExtractNN(s, U, k)
11     for each w ∈ U do
12         kNN(w) ← UpdateNN(kNN(w), v, s[w], k)
13     for each i = 1...m s.t. v[i] > 0 do
14         L_i ← L_i ∪ {(v, v[i])}
```

is that either $sim(\mathbf{v},\mathbf{w})$ or $sim(\mathbf{w},\mathbf{v})$ is computed, but not both of them as is the case for the basic inverted index-based algorithm. This is possible because $\mathbf{v}\cdot\mathbf{w} = \mathbf{w}\cdot\mathbf{v}$. As a result, the number of computations is divided by two.

However, dynamic index building cannot be straightforwardly applied to k-NN graph computation because the top-k extracted on Line 10 can only contain vectors that are already indexed. Hence, the top-k's computed on Line 10 are not complete. Therefore, our version also updates the existing top-k's (Lines 11-12) when new corresponding similarities have been computed. Furthermore, we mitigate the computational cost of performing these updates by storing the temporary top-k's as heap structures.

5.7 PrunedIndex

To further improve over DynamicIndex (Alg. 5), we adopt techniques proposed for solving the related problem called *all pairs similarity search* or *similarity join* which comprises finding all pairs of vectors that are more similar than a certain threshold [7, 4, 13, 6, 50]. However, these techniques heavily rely upon this threshold, which is absent in our case. Indeed, the similarity of the k-nearest neighbor can be close to zero for one vector while it can be close to one for another vector. Consider for example Figure 5.1, which shows for the *dblp* dataset the cumulative distribution of the similarity with the k-nearest neighbor. This Figure shows that the similarity with the k-nearest neighbor spans the whole domain, i.e. from 0 to 1, and that for 50% of the vectors, even the 1-nearest neighbor only has a similarity lower than 0.5. A user can, for example, search for the 3 papers most similar to a target paper, and not care that their absolute similarity is below a certain threshold.

To solve this problem, we propose PrunedIndex, a solution that consists of four

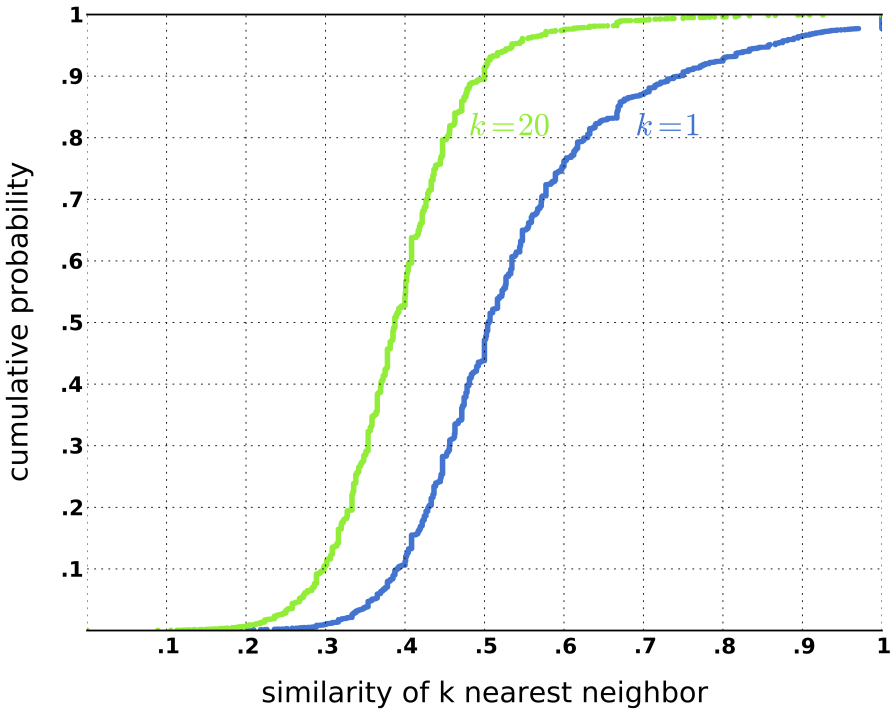

Figure 5.1: Cumulative distribution of the similarity with the k-nearest neighbor for all objects in the *dblp* dataset.

steps. In the first step, we introduce a virtual threshold τ and for each vector in $\mathcal{V}$ we efficiently search for the k-nearest neighbors that have a similarity larger than τ with this vector. After this first step, we only found the top-k's for which all members have a similarity above τ. Consequently, we did not yet compute all top-k's. Therefore, in the next three steps, we identify and compute the remaining top-k's.

Before discussing PrunedIndex in detail, we define two essential concepts: max($\mathbf{v}$) and $\max_i(\mathcal{V})$.

Definition Given a vector $\mathbf{v} \in \mathbb{R}^m$, $max(\mathbf{v})$ is the maximum value $\mathbf{v}[i]$ that appears in $\mathbf{v}$ over all dimensions $i \in [1, m] \subset \mathbb{N}$.

Definition Given a set of vectors $\mathcal{V} \subseteq \mathbb{R}^m_{\geq 0}$, $\max_i(\mathcal{V})$ is the maximum value $\mathbf{v}[i]$ that appears in dimension i over all vectors $\mathbf{v} \in \mathcal{V}$.

Algorithm 6 describes PrunedIndex. In the preprocessing step (Lines 1-4), the vectors v are sorted in descending order of max(v) and the dimensions i are re-ordered in descending order of $c(i)$. This is required by the optimization techniques. The remainder of the algorithm is organized in four steps: the thresholding step (Lines 5-10), the partitioning step (Lines 11-18), the leftovers step (Lines 19-20), and the completion step (Lines 21-24).

Algorithm 6: PrunedIndex

input : $\mathcal{V}$, k, τ
output : For every $\mathbf{v} \in \mathcal{V}$: $kNN(\mathbf{v})$

1 Reorder the dimensions $i = 1 \dots m$ in descending order of $c(i)$.
2 Sort all $\mathbf{v} \in \mathcal{V}$ in descending order of max($\mathbf{v}$).
3 $\forall L_i \in \mathcal{L} : L_i \leftarrow \{\}$.
4 $\forall \mathbf{v} \in \mathcal{V} : kNN(\mathbf{v}) \leftarrow$ *empty heap*

//Thresholding step
5 **for each** $\mathbf{v} \in \mathcal{V}$ **do**
6 $\quad \mathbf{s}, U \leftarrow ComputeSimilarities(\mathbf{v}, \mathcal{L}, \tau)$ ◁ Alg. 8
7 $\quad kNN(\mathbf{v}) \leftarrow ExtractNN(\mathbf{s}, U, k)$
8 $\quad$ **for each** $\mathbf{w} \in U$ **do**
9 $\quad\quad kNN(\mathbf{w}) \leftarrow UpdateNN(kNN(\mathbf{w}), v, \mathbf{s}[w], k)$
10 $\quad \mathcal{L} \leftarrow GrowIndex(\mathcal{L}, \mathbf{v}, \tau)$ ◁ Alg. 7

//Partitioning step
11 $C \leftarrow \{\}$
12 $I \leftarrow \{\}$
13 **for each** $\mathbf{v} \in \mathcal{V}$ **do**
14 $\quad$ **if** $|kNN(\mathbf{v})| \geq k$ **and** $\min(kNN(\mathbf{v})) \geq \tau$ **then**
15 $\quad\quad C \leftarrow C \cup \{\mathbf{v}\}$
16 $\quad$ **else**
17 $\quad\quad I \leftarrow I \cup \{\mathbf{v}\}$
18 $\quad\quad kNN(\mathbf{v}) \leftarrow$ *empty heap*

//Leftovers step
19 $\forall L_i \in \mathcal{L} : L_i \leftarrow \{\}$.
20 $\mathcal{L}$, For every $\mathbf{v} \in I$: $kNN(\mathbf{v}) \leftarrow DynamicIndex(I, k)$

//Completion step
21 **for each** $\mathbf{v} \in C$ **do**
22 $\quad \mathbf{s}, U \leftarrow ComputeSimilarities(\mathbf{v}, \mathcal{L}, 0)$
23 $\quad$ **for each** $\mathbf{w} \in U$ **do**
24 $\quad\quad kNN(\mathbf{w}) \leftarrow UpdateNN(kNN(\mathbf{w}), v, \mathbf{s}[w], k)$

PrunedIndex achieves its runtime reduction in the **thresholding step** (Lines 5-10). For each vector $\mathbf{v} \in \mathcal{V}$ we efficiently search for the other vectors $\mathbf{w}$ that belong to the top-k of $\mathbf{v}$ and additionally have a similarity with $\mathbf{v}$ that is larger than the virtual threshold τ. Thanks to the virtual threshold τ, we avoid to consider candidates that will not be in the top-k anyway. Unfortunately, by exploiting τ, we also filter out some eventual top-k members. Consequently, we do not have an exact solution yet after the first step. Therefore, we sacrifice a part of the runtime reduction achieved in this first step to execute the other three steps, which transform the incomplete solution into a complete one.

In the **partitioning step** (Lines 11-18), we identify the vectors with a completed

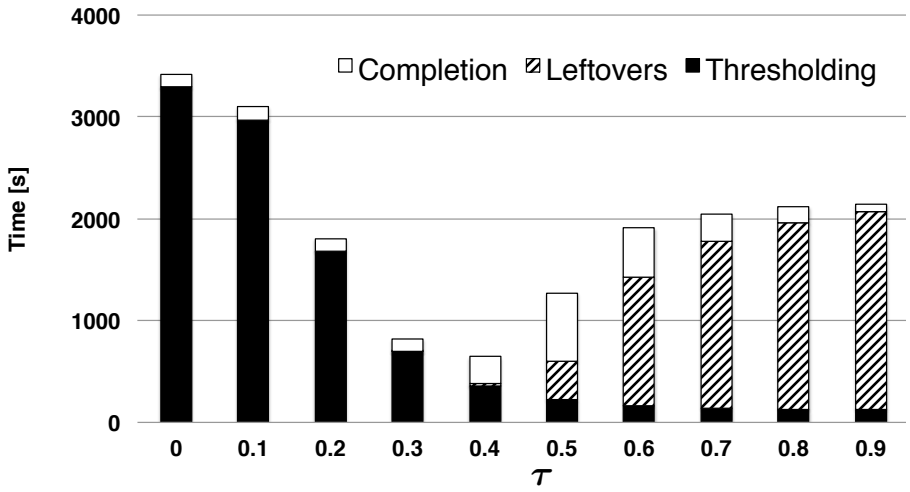

Figure 5.2: Time consumed by the different blocks of *PrunedIndex* for computing all 1-nearest neighbors on the *dblp* dataset for different settings of the virtual threshold τ.

top-k and put them in C. We retain the other vectors in I and empty their top-k.

In the **leftovers step** (Lines 19-20), we empty the inverted index $\mathcal{L}$ and compute all similarities among the vectors $\mathbf{v} \in I$ and the corresponding top-k's without any threshold. This is done by executing DynamicIndex (Alg. 5) with the subset $I \subseteq \mathcal{V}$ instead of the complete set of vectors $\mathcal{V}$ as input. Additionally, we also retain the inverted index $\mathcal{L}$ built in this step.

The vectors in C, on the other hand, are considered in the **completion step**. We compute the similarity of these vectors with all vectors in the inverted index, which is all vectors in I, to update the top-k's of these vectors. Since the top-k's of the vectors in C are already complete, we do not need to update and index them anymore.

The virtual threshold τ is set as high as possible to maximize pruning, but low enough to avoid too much incomplete neighborhoods after the thresholding step. Figure 5.2 shows the time consumed by every step of PrunedIndex for computing all 1-nearest neighbors on the *dblp* dataset for different settings of the virtual threshold τ. First, the preprocessing step (Lines 1-4) and the partitioning step (Lines 11-18), in which the incomplete top-k's are identified, consume a negligible amount of time and are therefore not visible on the figure. Second, as the virtual threshold τ is increased, more candidates are filtered out, and consequently the time consumed by the thresholding step (Lines 5-10) decreases rapidly. Unfortunately, also the number of actual top-k members that is filtered out increases. Consequently, more vectors will end-up in I and the time consumed by the leftovers step (Lines 19-20) monotonically increases with τ. Third, the time consumed by the completion step (Lines 21-24) first increases, reaches a maximum around $\tau = 0.5$, and then decreases again. The reason for the initial increase is that the size of I, and consequently the inverted index $\mathcal{L}$, which is an input for *ComputeSimilarities* on Line 22, is larger when τ is larger. The reason for the final decrease is that the number of items in C is smaller for a larger τ. The maximum is reached when the decreasing effect becomes

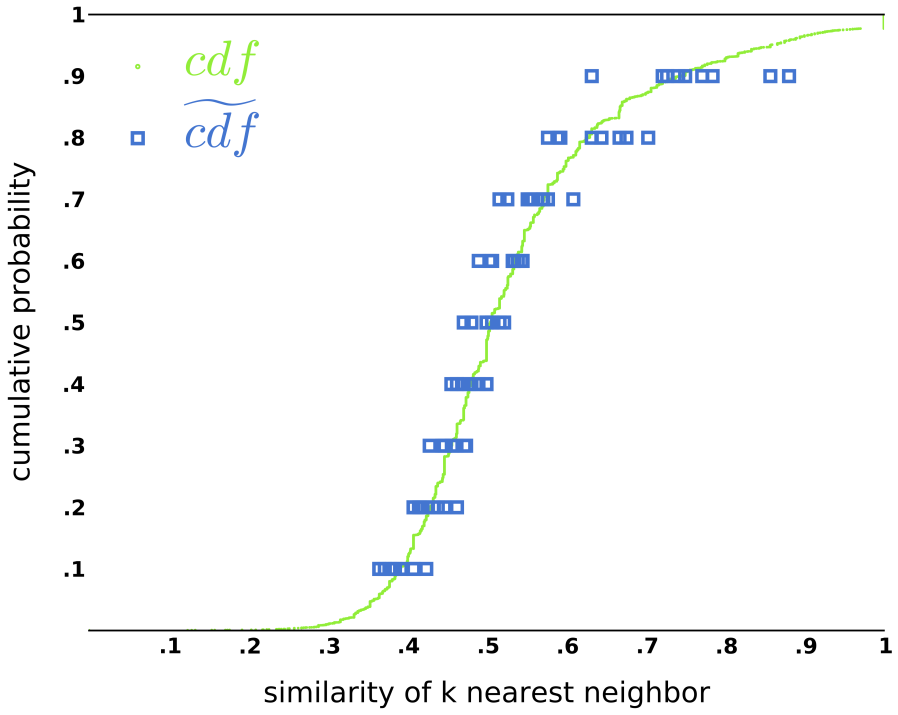

Figure 5.3: The cumulative distribution function *cdf* and 10 different approximations $\widetilde{cdf}$ based on 10 different random samples of size $|\mathcal{V}|/10^4$ from all $|\mathcal{V}|$ neighborhoods on the *dblp* dataset for $k = 1$.

more important than the increasing effect.

A good value for τ can be obtained by inspecting the cumulative distribution of the similarity with the k-nearest neighbor. Figure 5.1, for example, shows that $\tau = 0.4$ is a good setting on the *dblp* dataset when $k = 1$ because 90% of the neighborhoods is efficiently computed in the thresholding step, while only 10% is incomplete. Choosing τ smaller would sacrifice too much pruning power, while choosing τ larger would drastically increase the number of incomplete neighborhoods to 45% because of the steep gradient in the cumulative distribution. For $k = 20$, $\tau = 0.3$ is a good choice on the *dblp* dataset. However, this cumulative distribution is a result of PrunedIndex and can therefore not be used to set τ. Fortunately, the bottom part of the cumulative distribution can be robustly approximated by computing only 0.01% of the neighborhoods with DynamicIndex, which only takes slightly more than 0.01% of the total time. This is illustrated for the *dblp* dataset and $k = 1$ in Figure 5.3. In our experiments (Sec.5.9), we achieved good results by automatically choosing τ such that $\bar{cdf}(\tau) = 0.1$ with $\bar{cdf}$ the empirical approximation of the cumulative distribution by $|\mathcal{V}|/10^4$ random neighborhoods.

In the thresholding step, the runtime reduction is achieved through exploiting the virtual threshold τ in the functions *ComputeSimilarities* (Line 6) and *GrowIndex* (Line 10). We discuss them in the next two subsections.

5.7.1 GrowIndex

Instead of adding complete vectors to the inverted index, as done in DynamicIndex (Alg. 5, Lines 13-14), we only index a part of every vector, as proposed by Bayardo et al. [7]. As such, the size of the inverted index can be drastically reduced. This results into two advantages. First, the smaller index can be traversed faster. Second, less candidates for top-k membership will be generated later on, which reduces the time required to choose the top-k among them.

More specifically, we will avoid indexing a prefix of every vector $\mathbf{v}$, denoted $\mathbf{v}'_p$, with the highest non-zero dimension in the prefix smaller than or equal to p. Hence, $\mathbf{v}'_p \in \mathbb{R}^m_{\geq 0}$ and for all dimensions $j > p$, $\mathbf{v}'_p[j] = 0$. Furthermore, the remaining suffix of $\mathbf{v}$ that is added to the index is denoted $\mathbf{v}''_p$. Notice that the equalities $\mathbf{v} = \mathbf{v}'_p + \mathbf{v}''_p$ and $\mathbf{v} \cdot \mathbf{w} = \mathbf{v}'_p \cdot \mathbf{w} + \mathbf{v}''_p \cdot \mathbf{w}$ are a trivial consequence of these definitions. Practically, we only keep the sparse representation of $\mathbf{v}'_p$ and the inverted index $\mathcal{L}$ in memory. Before the algorithm starts, no vectors are indexed yet and all information is contained in the non-indexed parts $\mathbf{v}'_p$ with $p = m$. As the algorithm gradually indexes vectors, pairs $(i, \mathbf{v}[i])$ are removed from $\mathbf{v}'_p$ and inserted into L_i as $(v, \mathbf{v}[i])$. Consequently, p becomes smaller than m.

We can safely index only a part of every vector by exploiting two upper bounds on $sim(\mathbf{v}'_p, \mathbf{w})$, the similarity between the non-indexed $\mathbf{v}'_p$ and any other vector $\mathbf{w}$ that will be processed and indexed after $\mathbf{v}$. The first upper bound was proposed by Bayardo et al. [7] and is based on the definition of the dotproduct:

$$sim(\mathbf{v}'_p, \mathbf{w}) = \mathbf{v}'_p \cdot \mathbf{w} = \sum_{i=1}^{p} \mathbf{v}[i] \cdot \mathbf{w}[i], \tag{5.1}$$

in which $\mathbf{w}[i]$ has two upper bounds. The first one is $\max_i(\mathcal{V})$ and is related to the dimension i. The second one is $\max(\mathbf{v})$ and is a result of the ordering of the vectors according to their maximum value (Line 2 of PrunedIndex). Consequently, the first upper bound on $sim(\mathbf{v}'_p, \mathbf{w})$ is

$$\begin{aligned} sim(\mathbf{v}'_p, \mathbf{w}) &\leq \sum_{i=1}^{p} \mathbf{v}[i] \cdot \min\left(\max_i(\mathcal{V}), \max(\mathbf{v})\right). \\ &\leq b_1 \end{aligned} \tag{5.2}$$

The second upper bound was proposed by Anastasiu and Karypis [4] and is based on the Cauchy-Schwartz inequality:

$$sim(\mathbf{v}'_p, \mathbf{w}) = \mathbf{v}'_p \cdot \mathbf{w} \leq ||\mathbf{v}'_p||_2 \cdot ||\mathbf{w}||_2,$$

which reduces to the second upper bound on $sim(\mathbf{v}'_p, \mathbf{w})$:

$$\begin{aligned} sim(\mathbf{v}'_p, \mathbf{w}) &\leq ||\mathbf{v}'_p||_2, \\ &\leq b_2 \end{aligned} \tag{5.3}$$

since all vectors are unit length normalized.

To maximize the runtime reduction, we maximize the size of the prefix $\mathbf{v}'_p$ that we can avoid to index. This maximum is determined by the function *ComputeSimilarities*, which we will explain in detail in Section 5.7.2. To make sure that *ComputeSimilarities* can identify all top-k members as candidates, it is required that

for every vector $\mathbf{w} \in \mathcal{V}$ that is not yet indexed, and for which $sim(\mathbf{v},\mathbf{w}) \geq \tau$, we index at least one non-zero value $\mathbf{v}[i]$ for which also $\mathbf{w}[i] > 0$. Informally, we say that we need to index at leat one "match" between $\mathbf{v}$ and $\mathbf{w}$. Bayardo et al. [7] prove that this is a sufficient condition for producing correct results. Algorithm 7 describes how *GrowIndex* realizes this objective by exploiting the upper bounds b_1 and b_2. GrowIndex iterates over the non-zero dimensions of $\mathbf{v}$ (Line 5) and updates the upper bounds b_1 and b_2 on $sim(\mathbf{v}'_i,\mathbf{w})$ every time the dimension i is increased (Lines 6-8). Now, as long as the minimum of both upper bounds is lower than τ, obviously also $sim(\mathbf{v}'_i,\mathbf{w})$ is lower than τ and there are two options: either there exists a dimension $j > i$ for which both $\mathbf{v}[j] > 0$ and $\mathbf{w}[j] > 0$, i.e. there is a match in a higher dimension, or there isn't. On the one hand, if there is a match in a dimension $j > i$, then *ComputeSimilarities* can use this match to identify all top-k members and we do not need to index $\mathbf{v}[i]$. On the other hand, if there isn't, $sim(\mathbf{v}'_i,\mathbf{w})$ is equal to $sim(\mathbf{v},\mathbf{w})$ and since $sim(\mathbf{v}'_i,\mathbf{w}) \leq \min(b_1,b_2) < \tau$, also $sim(\mathbf{v},\mathbf{w})$ will be smaller than τ and we are not interested in finding this similarity. Consequently, also in the second case we do not need to index $\mathbf{v}[i]$.

GrowIndex requires that all $\mathbf{v} \in \mathcal{V}$ are sorted in descending order of $\max(\mathbf{v})$ (Line 2 in Alg. 6). In this way, the upper bound on the value $\mathbf{w}[i]$ in Equation 5.1 can be set to $\min(\max_i(\mathcal{V}), \max(\mathbf{v}))$ (Eq. 5.2), instead of the looser bound $\max_i(\mathcal{V})$.

Although *GrowIndex* does not require any specific ordering of the dimensions $i = 1,\ldots,m$, we follow Bayardo et al. and order them in descending order of $c(i)$ to heuristically minimize the maximum length of all inverted lists (Line 1 in Alg. 6).

Algorithm 7: GrowIndex

input : $\mathcal{L}$, $\mathcal{V}$, $\mathbf{v} \in \mathcal{V}$, $\tau \in [0,1] \subset \mathbb{R}$
output: $\mathcal{L}$

1 $p \leftarrow m$
2 $\mathbf{v}'_p \leftarrow \mathbf{v}$
3 $\mathbf{v}''_p \leftarrow \mathbf{0}_{m\times 1}$
4 $b_1, b_2, b_t \leftarrow 0$
5 **for each** $i = 1 \ldots m$ **s.t.** $\mathbf{v}[i] > 0$ **do**
6 $\quad b_1 \leftarrow b_1 + \mathbf{v}[i] \cdot \min(\max_i(\mathcal{V}), \max(\mathbf{v}))$
7 $\quad b_t \leftarrow b_t + \mathbf{v}[i]^2$
8 $\quad b_2 \leftarrow \sqrt{b_t}$
9 $\quad$ **if** $\min(b_1,b_2) \geq \tau$ **then**
10 $\quad\quad p \leftarrow \min(p,i)$
11 $\quad\quad L_i \leftarrow L_i \cup \{(v, \mathbf{v}[i], ||\mathbf{v}'_i||_2)\}$
12 $\quad\quad \mathbf{v}'_p[i] \leftarrow 0$
13 $\quad\quad \mathbf{v}''_p[i] \leftarrow \mathbf{v}[i]$

5.7.2 ComputeSimilarities

Instead of processing every relevant inverted list, as is done in DynamicIndex (Alg. 8, Lines 4-9), *ComputeSimilarities* (Alg. 8) reduces the number of inverted lists that needs to be scanned and does not necessarily scan the remaining relevant inverted

list completely. In this way, the scanning itself takes less time, but also less candidate top-k members are generated, i.e. $|U|$ is kept smaller. Furthermore, *ComputeSimilarities* contains additional steps (Alg. 8, Lines 19-23) for correctly dealing with the partial inverted index created by *GrowIndex*. We first discuss these additional steps and afterwards four optimizations.

When computing the k-nearest neighbors of a vector $\mathbf{v}$, *ComputeSimilarities* computes a score $\mathbf{s}[w]$ for every partially indexed vector $\mathbf{w}$ by comparing $\mathbf{v}$ with the partial inverted index $\mathcal{L}$ (Lines 5-18). This score is equal to the dotproduct of $\mathbf{v}$ with the indexed part of $\mathbf{w}$, i.e. $\mathbf{s}[w] = \mathbf{v} \cdot \mathbf{w}''_p$. Hence, by adding the dotproduct $\mathbf{v} \cdot \mathbf{w}'_p$ to this score (Line 23), it becomes equal to $\mathbf{v} \cdot \mathbf{w}$, which is the exact similarity between $\mathbf{v}$ and $\mathbf{w}$. Explicitly computing the dotproduct $\mathbf{v} \cdot \mathbf{w}'$ (Line 23) is an expensive operation. This partially offsets the runtime reduction achieved by only indexing the minimally required parts of the vectors in *GrowIndex*.

The first optimization is to avoid this explicit dotproduct computation $\mathbf{v} \cdot \mathbf{w}'_p$ (Line 23) as much as possible. This is possible by exploiting an upper bound on $\mathbf{v} \cdot \mathbf{w}'_p$ (Line 20) that was proposed by Bayardo et al. [7]. The upper bound is based on the definition of the dotproduct

$$\mathbf{v} \cdot \mathbf{w}'_p = \sum_{i=1}^{p} \mathbf{v}[i] \cdot \mathbf{w}'_p[i].$$

In this definition, we can replace every factor by an upper bound to obtain an upper bound on $\mathbf{v} \cdot \mathbf{w}'_p$

$$\mathbf{v} \cdot \mathbf{w}'_p \leq \sum_{i=1}^{p} \max(\mathbf{v}) \cdot \max(\mathbf{w}'_p),$$

which can be tightened to

$$\mathbf{v} \cdot \mathbf{w}'_p \leq \min(|\mathbf{v}|, |\mathbf{w}'_p|) \cdot \max(\mathbf{v}) \cdot \max(\mathbf{w}'_p). \tag{5.4}$$

Hence, if

$$\mathbf{s}[w] + \min\left(|\mathbf{v}|, |\mathbf{w}'_p|\right) \cdot \max(\mathbf{v}) \cdot \max(\mathbf{w}'_p) < \tau,$$

also

$$\mathbf{s}[w] + \mathbf{v} \cdot \mathbf{w}'_p < \tau,$$

which is equivalent to

$$\mathbf{v} \cdot \mathbf{w} < \tau,$$

and we do not need to compute the dotproduct exactly, which is expensive.

The second optimization exploits an upper bound on $\mathbf{v} \cdot \mathbf{w}$, proposed by Bayardo et al., that is again derived from the definition of the dotproduct:

$$sim(\mathbf{v}, \mathbf{w}) = \sum_{i=1}^{m} \mathbf{v}[i] \cdot \mathbf{w}[i],$$

and again we obtain an upper bound on the dotproduct by replacing every value with an upper bound:

$$sim(\mathbf{v}, \mathbf{w}) \leq \sum_{i=1}^{m} \max(\mathbf{v}) \cdot \max(\mathbf{w}),$$

which we can tighten to

$$sim(\mathbf{v},\mathbf{w}) \leq |\mathbf{w}| \cdot \max(\mathbf{v}) \cdot \max(\mathbf{w}),$$

and simplify to

$$sim(\mathbf{v},\mathbf{w}) \leq |\mathbf{w}| \cdot \max(\mathbf{v}) \cdot 1,$$

since all vectors are normalized to have unit length. Hence, if the upper bound $|\mathbf{w}| \cdot \max(\mathbf{v})$ is smaller than τ, also $sim(\mathbf{v},\mathbf{w})$ is smaller than τ and we do not need to consider $\mathbf{w}$ as a candidate neighbor for $\mathbf{v}$, given the threshold τ. Equivalently, we only consider vectors $\mathbf{w}$ for which $|\mathbf{w}| > \tau / \max(\mathbf{v})$. Since all vectors $\mathbf{v} \in \mathcal{V}$ are ranked in descending order of $\max(\mathbf{v})$ (Line 2), the lower bound $\tau / \max(\mathbf{v})$ on $|\mathbf{w}|$ becomes tighter as the algorithm progresses. Therefore, we can permanently remove from the index any vector w for which $|\mathbf{w}| < \sigma$. *ComputeSimilarities* removes such vectors only if they are at the beginning of an inverted list L_i that needs to be traversed (Lines 6-9) to avoid that overhead computations will completely offset the achieved optimization. Consequently, it is possible that there are vectors $\mathbf{w}$ further down L_i that could be removed but are not. However, this will only be exceptions because one typically observes that $|\mathbf{w}|$ becomes smaller as one moves towards the beginning of the inverted lists. This is because the vectors $\mathbf{w}$ are added to the inverted index in decreasing order of $\max(\mathbf{w})$, and $\max(\mathbf{w})$ is inversely correlated with $|\mathbf{w}|$ because of the normalization to unit norm (Sec. 5.2).

In our implementation, we do not explicitly remove $(f, \mathbf{f}[i])$ from memory but maintain an offset from which to start the list iteration instead. Explicitly removing $(f, \mathbf{f}[i])$ from the beginning of L_i would be expensive since it would require shifting all elements in L_i because we use a compact data structure, such as an array in C++, to store L_i in main memory. In a compact data structure, consecutive values in the data structure are stored in memory locations with consecutive addresses. This is for example not the case for a standard linked list. As a result, we can benefit from the cpu caching to more rapidly traverse the inverted lists L_i. In our experiments, we noticed that the benefit of this implementation detail, i.e. using compact data structures for the inverted lists, was dramatic, which is in line with the observations of Stroustrup [91]. If memory size became an issue, an explicit removal of all elements before the offset could be periodically organized to reclaim the corresponding memory.

The third optimization exploits the upper bound in Equation 5.3, derived from the Cauchy-Schwarz inequality. Now, if $\mathbf{w}[i] = 0$ for every $i > p$, $||\mathbf{v}'_p||_2$ is not only an upperbound on $sim(\mathbf{v}'_p, \mathbf{w})$ but also on $sim(\mathbf{v},\mathbf{w})$. Hence, if $||\mathbf{v}'_p||_2 < \tau$ and $\mathbf{w}[i] = 0$ for every $i > p$, also $sim(\mathbf{v},\mathbf{w}) < \tau$ and we can discard the candidate $\mathbf{w}$. This is exploited on Line 11 of *ComputeSimilarities*, taking into account that the condition $\mathbf{s}[w] = 0$ while iterating from dimension m to dimension 1 (Line 5) is equivalent to the condition that $\mathbf{w}[i] = 0$ for every $i > p$.

The fourth and final optimization, applied on Line 13 of *ComputeSimilarities*, was proposed by Anastasiu and Karypis [4] and directly exploits the Cauchy-Schwarz inequality:

$$\mathbf{v}'_{i-1} \cdot \mathbf{w}'_{i-1} \leq ||\mathbf{v}'_{i-1}||_2 \cdot ||\mathbf{w}'_{i-1}||_2. \tag{5.5}$$

Therefore, if $\mathbf{s}[w] + ||\mathbf{v}'_{i-1}||_2 \cdot ||\mathbf{w}'_{i-1}||_2$ is smaller than τ, also $\mathbf{s}[w] + \mathbf{v}'_{i-1} \cdot \mathbf{w}'_{i-1}$ will be smaller than τ. Furthermore, since $\mathbf{s}[w] = \mathbf{v}''_{i-1} \cdot \mathbf{w}''_{i-1}$, also $sim(\mathbf{v},\mathbf{w})$ will be smaller than τ and we can discard $\mathbf{w}$ as a candidate (Lines 14-15).

For the all pairs similarity search problem, various other optimizations where proposed[7, 4, 13, 6, 50]. The effectiveness of an optimization depends on the tightness of the involved bounds, the cost of computing the bounds, their dependence on τ and their complementarity with the other optimizations that are used. Overall, we found that the particular combination of optimization proposed above, works best in our problem setting.

Algorithm 8: ComputeSimilarities

input : $\mathcal{L}$, $\mathbf{v} \in \mathcal{V}$, $\tau \in [0,1] \subset \mathbb{R}$
output: $\mathbf{s} \in \mathbb{R}^{|\mathcal{V}|}$: $s[w] = sim(\mathbf{v},\mathbf{w})$ for all $\mathbf{w} \in \mathcal{V}$ and,
$U = \{\mathbf{u} \in \mathcal{V} \mid \mathbf{s}[u] > 0\}$

$\mathbf{s} \leftarrow \mathbf{0}_{|\mathcal{V}| \times 1}$
$U \leftarrow \{\}$
$\rho \leftarrow 1$
$\sigma \leftarrow \tau / \max(\mathbf{v})$
for each $i = m \ldots 1$ **s.t.** $\mathbf{v}[i] > 0$ **do**
  $(f, \mathbf{f}[i]) \leftarrow$ first pair in L_i
  while $|\mathbf{f}| < \sigma$ **do**
    remove $(f, \mathbf{f}[i])$ from L_i
    $(f, \mathbf{f}[i]) \leftarrow$ new first pair in L_i
  for each $(w, \mathbf{w}[i]) \in L_i$ **do**
    if $\mathbf{s}[w] > 0$ **or** $\rho \geq \tau$ **then**
      $\mathbf{s}[w] \leftarrow \mathbf{s}[w] + \mathbf{v}[i] \cdot \mathbf{w}[i]$
      if $\mathbf{s}[w] + ||\mathbf{v}'_{i-1}||_2 + ||\mathbf{w}'_{i-1}||_2 < \tau$ **then**
        $\mathbf{s}[w] \leftarrow 0$
        $U \leftarrow U \setminus \{\mathbf{w}\}$
      else
        $U \leftarrow U \cup \{\mathbf{w}\}$
  $\rho \leftarrow ||\mathbf{v}'_{i-1}||_2$
for each $\mathbf{w} \in U$ **do**
  if $\mathbf{s}[w] + \min\left(|\mathbf{v}|, |\mathbf{w}'_p|\right) \cdot \max(\mathbf{v}) \cdot \max(\mathbf{w}'_p) < \tau$ **then**
    $U \leftarrow U \setminus \{\mathbf{w}\}$
  else
    $\mathbf{s}[w] \leftarrow \mathbf{s}[w] + \mathbf{v} \cdot \mathbf{w}'_p$

5.8 Related work

A first line of related research is concerned with answering k-NN queries as fast as possible. To this end, most of the methods in this category rely on pre-built data structures. The time for building these structures is often neglected since it is assumed that the amount of queries will be so high that the total time to answer all

queries will exceed the time for building the data structure by far. Furthermore, it is also assumed that the data structure can be built "off-line", i.e. with no strict speed requirements, whereas the queries need to be answered "on-line", i.e. an immediate answer is expected. Some well known algorithms in this domain are kd-tree [9, 26], k-means-tree [27], PCA-tree [86], vp-tree [115] and ball-tree [51]. Unfortunately, the performance of these methods rapidly decreases for high dimensional data [59, 107].

Consequently, this line of research also contains approximate algorithms that sacrifice accuracy to achieve runtime reduction. Muja and Lowe [59] provide an up-to-date overview of the work in this domain. They classify the approximate algorithms in three categories: partitioning trees, hashing techniques and neighboring graph techniques. Although approximate algorithms are required for datasets above a certain size in order to finish in a reasonable amount of time, exact solutions are obviously preferred if their runtime is not too large.

Therefore, this line of research also contains algorithms that speedup the exact computation of the k-nearest neighbors for medium dimensional data such that exact solutions can be computed efficiently for bigger datasets [107, 40]. However, all these methods build a data structure (e.g. a tree or an inverted index) completely in advance because they need to be able to answer every possible query. Thus, when used for finding all exact k-nearest neighbors, they compute every similarity $sim(v,w) = sim(w,v)$ twice, which heavily hinders performance. For constructing the k-nearest neighbors graph however, it is not required that every possible query can be answered.

The second line of research, on the other hand, is specifically concerned with efficiently constructing the k-nearest neighbors graph (or performing a k-nearest neighbors self-join). Therefore, it explicitly takes advantage of the fact that in this case not all possible queries need to be answered. Furthermore, it contains both exact [69, 16, 116] and approximate [14, 23, 70] algorithms. The approximate algorithms, on the one hand, have been shown to work for high-dimensional data [14, 23, 70]. The exact algorithms, on the other hand, have only been demonstrated on low-dimensional data [69, 16]. The limitation of exact algorithms to low-dimensional data does not come as a surprise. However, this limitation could be (partially) removed for sparse data by exploiting their sparseness. Our algorithms do exactly that.

There does already exists a collection of algorithms that takes advantage of sparseness in the data. These are the Top-k document retrieval algorithms, since every document only contains a small portion of the total vocabulary. Searching all exact k-nearest neighbors is technically possible with top-k document retrieval algorithms, which forms a third line of related research. In order to do so, one first needs to build an index with all vectors $v \in \mathcal{V}$, which serve as "documents". Secondly, one needs to use the same vectors $v \in \mathcal{V}$ as "queries" and retrieve for every "query" the top-k "documents" from the index. For this problem, optimizations exist already for two decades [97] and studying even better optimizations has been the subject of more recent research [43, 90, 22]. However, as we show in our experimental evaluation (Sec. 5.9), these algorithms cannot efficiently find all exact k-nearest neighbors. There are two reasons for this. First, also document retrieval algorithms compute both $sim(v,w)$ and $sim(w,v)$, which obviously hinders performance. Second, document retrieval algorithms achieve runtime reductions through skipping inverted lists that easily contain more than 10 000 elements. By skipping large parts of the

enormous inverted lists used in typical document retrieval applications they avoid many disk access operations and/or decoding operations. However, a typical inverted list considered in this chapter easily contains less than 100 elements, which is 100 times less. Furthermore, typical keyword queries in document retrieval contain approximately 2 words. Our "queries", on the other hand, are easily 10 times longer. As a result, the number of candidate top-k members is often larger than the inverted lists on our datasets. Consequently, it is not possible to skip large parts of an inverted list.

Datasets more similar to ours are considered by the fourth line of related research: the all-pairs similarity search problem, also called the similarity-join problem or ϵ-nearest neighbors problem[7, 4, 13, 6, 50]. However, in this case, the goal is to find all pairs of vectors for which the similarity is higher than a certain threshold. Obviously, all efficient solutions to this problem exploit this threshold. Exactly due to the absence of this threshold in our problem setting, it is much less obvious to efficiently solve it. One of our solutions is *PrunedIndex*, introduces a virtual threshold that allows us to adopt the optimization techniques from the all-pairs similarity search problem.

A final related problem setting that is very similar to the similarity-join is the top-k similarity-join [112]. In this case, the goal is to find the k most similar pairs of vectors in a collection. This problem is very similar to the ordinary similarity-join because in both cases there is a global threshold. For ordinary similarity-join, this global threshold is fixed. For top-k similarity-join this global threshold increases as the global top-k is updated. Despite the phrase "top-k" in the name, this is very different from our problem setting in which there exists no global threshold as we want to find the local top-k for every individual vector.

5.9 Runtime Comparison

We compared our proposed algorithms, *DynamicIndex* and *PrunedIndex* with the *Naïve Baseline* outlined in Algorithm 2, the *Basic Inverted Index*-based algorithm (Alg. 4), and two often used top-k document retrieval algorithms. Top-k document retrieval algorithms are the only state-of-the-art algorithms that were already demonstrated to produce exact results in a reasonable amount of time on high dimensional data. Since other exact algorithms assume dense data, therefore do not exploit any sparseness, and were only demonstrated on maximally, 3 [16], 16 [116], and 24 [69] dimensions, which is much less than the more than 700 000 dimensions in the *dblp* dataset, they were not considered in our comparison.

The first document retrieval algorithm is often referred to as *max-score* and is a well known document-at-a-time approach. It was first described by Turtle and Flood [97], but we use the more recent version described by Jonassen and Bratsberg [43]. The second one, denoted *SC* , originates from the work of Strohman and Croft [90] and is a well known term-at-a-time approach that uses a layered inverted index. We use the version described by Ding and Suel [22].

Table 5.3 summarizes the runtimes, in seconds, of the selected algorithms on the three datasets for two values of k, i.e. 1 and 20. From this table we make several observations. First, the *Naïve Baseline* is indeed very naïve. Second, our novel

Table 5.3: Runtimes of selected algorithms, in seconds, on multiple datasets and with multiple values of k. Lower is better. Best performances are in blue and underlined. Italic numbers are extrapolations from experiments that were stopped after running for more than 24 hours.

		PrunedIndex	*DynamicIndex*	*Basic Inverted Index*	*Naïve Baseline*
dblp	$k = 1$	648	1 981	4 108	26 260
	$k = 20$	972	1 950	4 115	26 007
msd	$k = 1$	3 288	2 945	7 091	$2.5 \cdot 10^6$
	$k = 20$	3 931	2 964	7 107	$2.5 \cdot 10^6$
tweets	$k = 1$	1 265	3 517	10 836	54 045
	$k = 20$	1 687	3 577	10 685	53 738

algorithms, PrunedIndex and DynamicIndex, are the fastest on all three real world datasets and up to 8 times better than the *Basic Inverted Index*-based approach.

Third, the *PrunedIndex* algorithm performs better for $k = 1$ than it does for $k = 20$. This is not surprising given Figure 5.1: with $\tau = 0.4$, it is possible to finish 90% of the 1-nearest neighbors in the thresholding step, but only 50% of the 20-nearest neighbors can be completed in thresholding step. Consequently, the consecutive steps (partitioning, leftovers, and completion) take more time for $k = 20$.

Fourth, on the *dblp* and *tweets* datasets *PrunedIndex* outperforms *DynamicIndex*. On the *msd* dataset, it is the other way around. The key dataset property that determines this difference is $\max_{i=1\ldots m}\left(\frac{c(i)}{|\mathcal{V}|}\right)$, the maximum relative length of a dimension, summarized in Table 5.1. If it is higher, the pruning strategies of *PrunedIndex* work better. To support this argument, we create two artificial datasets. The first artificial dataset, *tweets*$^-$, is constructed by removing the longest dimension from the *tweets* dataset. From Table 5.4, we see that the advantage of *PrunedIndex* over *DynamicIndex* on *tweets* almost completely vanishes on *tweets*$^-$ as the maximum relative length of a dimension reduces from 0.44 to 0.13. The second artificial dataset, *msd*$^+$, is constructed by adding two new dimensions, $m_{msd}+1$ and $m_{msd}+2$, to *msd* . If user $\mathbf{v}$ is male, $\mathbf{v}[m_{msd}+1] = 1$ and $\mathbf{v}[m_{msd}+2] = 0$. If a user is female, $\mathbf{v}[m_{msd}+1] = 0$ and $\mathbf{v}[m_{msd}+2] = 1$. We randomly assigned a gender to every user with uniform probability. From Table 5.4, we see that by simply adding gender features, and correspondingly increasing $\max_{i=1\ldots m}\left(\frac{c(i)}{|\mathcal{V}|}\right)$ from 0.11 to 0.5, we created a dataset on which *PrunedIndex* is much faster than *DynamicIndex*, although it was slower before the gender features were added. As a rule of thumb, we can say that PrunedIndex is to be preferred over DynamicIndex if $\max_{i=1\ldots m}\left(\frac{l(i)}{|\mathcal{V}|}\right) \geq 0.3$.

Finally, we mention that the top-k document retrieval algorithms, *SC* and *max-score* , did not perform significantly better than the *Naïve Baseline*. This is because there are key differences between k-nearst neighbors graph construction and top-k document retrieval, as explained in Section 5.8.

Table 5.4: Basic properties of the datasets.

	runtime[s], $k = 1$		
	PrunedIndex	*DynamicIndex*	$\max_{i=1\ldots m}\left(\frac{c(i)}{\lvert\mathcal{V}\rvert}\right)$
tweets$^-$	1 049	1 288	0.13
msd$^+$	5 212	8 175	0.5

5.10 Conclusions

We proposed two novel algorithms to efficiently compute the exact k-nearest neighbors graph on high dimensional, sparse vector data. They are more than two times as fast as the current state-of-the-art.

Specializing the *PrunedIndex* algorithm for binary data, to further increase the speedup, is an interesting direction for future work.

CHAPTER 6

Conclusions

In this thesis we discussed collaborative filtering with binary, positive-only data. This problem is defined as scoring missing edges in an unweighted, bipartite graph according to their probability of existing in the future. Binary, positive-only data is typically associated with implicit feedback such as items bought, videos watched, songs listened to, books checked out from a library, ads clicked on, etc. However, it can also be the result of explicit feedback such as *likes* on social networking sites. While the obvious applications of collaborative filtering involve users and items, its relevance goes well beyond. It can, for example, connect photos with tags, or active drug compounds with biological targets.

6.1 Main Contributions

The main contributions of this thesis can be summarized as follows.

- We have presented a comprehensive survey of collaborative filtering methods for binary, positive-only data. The backbone of our survey is an innovative, unified matrix factorization perspective on collaborative filtering methods, also those that were typically not associated with matrix factorization models such as nearest neighbors methods and association rule mining-based methods. From this perspective, a collaborative filtering algorithm consists of three building blocks: a matrix factorization model, a deviation function and a numerical minimization algorithm. By comparing methods along these three dimensions, we were able to highlight surprising commonalities and key differences.

- We proposed KUNN, a novel method for collaborative filtering. KUNN originates from a reformulation that unifies user- and item-based nearest neighbors methods with analytically solvable deviation functions. Thanks to this reformulation, it becomes clear that user- and item-based nearest neighbors methods discard important parts of the available information. KUNN improves upon the existing nearest neighbors methods by actually using more of the available information. Our experimental evaluation shows that KUNN not only outperforms existing nearest neighbors methods, but also state-of-the-art matrix factorization methods.

- We challenged the well accepted belief that item-based neighborhood methods are superior for explaining the recommendations they produce. Thanks to our reformulation, we were able to show that also recommendations by KUNN and the traditional user-based neighborhood methods come with a natural explanation.

- We showed that the widely used item-based neighborhood methods fail when they make recommendations for shared accounts. Specifically, we identified the generality, dominance and presentation problems. As a first step towards solutions, we formally introduced the challenge of top-N recommendation for shared accounts in the absence of contextual information. Furthermore, we proposed the DAMIB-COVER algorithm, our solution to this challenge. Central to our approach, we showed a theorem that allowed us to compute a recommendation score in $\mathcal{O}\left(n \log n\right)$ instead of exponential time. Finally, we experimentally validated that our proposed solution has important advantages.

- We proposed two novel inverted index-based algorithms to efficiently compute the exact k-nearest neighbors graph on high dimensional, sparse vector data. They are up to eight times as fast as the current state-of-the-art. The key ideas behind our approach are to exploit the sparseness of the data and to relate the problem to the all-pairs similarity search problem via the introduction of a virtual threshold.

6.2 Outlook

The *Conclusions* sections of Chapters 2-5 outline very specific opportunities for future work related to those chapters. In this section, we provide a more high level, personal opinion on the opportunities in recommendation research.

First, the rating prediction problem is well researched, and its relevance will probably decline further in the future. Therefore, it does not seem a very exciting topic for future work. Binary, positive-only data, on the other hand, seems to be a good default assumption for future work. Although also relevant, less work exists on positive-only data that is not binary. Examples of such data are minutes watched, times played, time spent, etc. It would be interesting to study the implications of using such non-binary data and devise novel approaches that take these implications into account. An even more interesting, more relevant and less explored problem setting also takes into account the impression history of recommendations. In

this way, one also needs to deal with negative and temporal data, which drastically complicates the problem and its solutions.

Second, in this thesis, we applied the best practices concerning the experimental evaluation of our own and competitive methods on historical data. However, these evaluations are necessarily biased towards the known preferences of the users. The best way to evaluate collaborative filtering methods, is by means of A/B testing in real life environments. Although there already exist preliminary work, more work on the correlation between A/B testing results and evaluations on historical data is required to allow the field to advance in the right direction.

Third, to drastically improve the performance of collaborative filtering methods, models need to become richer, which implies that drastically more data (than typically used in academic experiments) is required to train these models without overfitting. Matrix factorization models have one set of latent dimensions in which both users and items are profiled. In other domains, models with one set of latent dimensions have been successfully enriched by adding more layers of latent dimensions. As such, not only users and items are profiled as vectors in latent dimensions, but these latent dimensions are in turn profiled as vectors in a second layer of latent dimensions etc. Whether such a strategy will proof useful for collaborative filtering is an exciting research question.

Finally, collaborative filtering research is mainly concerned with maximally exploiting the available information in order to provide the best possible recommendations, immediately. However, it is probably beneficial to also present recommendations that provoke maximally informative user feedback. As such, the best possible recommendations can be provided in the long run. Obviously, recommendations presented to maximally learn from the feedback cannot be totally irrelevant, and a balance between explore and exploit needs to be found. Most work in this area is related to the cold-start problem, when there is little or none information to exploit. The principle, however, is also applicable to mature users and items. Also on this aspect little work has been done, and exciting work still needs to be performed.

Nederlandse Samenvatting

Steeds meer mensen worden overweldigd door een overvloed aan keuzes. Via het World Wide Web heeft iedereen toegang tot een grote verscheidenheid aan nieuws, opinies, (encyclopedische) informatie, sociale contacten, boeken, muziek, films, foto's, producten, vakantiebestemmingen, jobs, huizen en vele andere items, en dit van over de hele wereld. Vanuit het oogpunt van één bepaalde persoon is de grote meerderheid van deze items echter niet interessant, en zijn de weinige interessante items begraven onder een gigantische berg oninteressante items. Er bestaan bijvoorbeeld veel boeken die ik met plezier zou lezen, tenminste als ik ze zou kennen. Daarnaast zijn er zelfs veel niche items die niet worden gemaakt omdat men anticipeert dat het doelpubliek ze niet zal vinden tussen de overvloed aan andere items. Bepaalde boeken bijvoorbeeld, worden nooit geschreven omdat hun schrijvers verwachten dat ze niet in staat zullen zijn een voldoende groot deel van hun doelpubliek te bereiken, hoewel dat doelpubliek wel degelijk bestaat.

Aanbevelingssystemen dragen bij aan de oplossing voor deze problemen, door individuen te verbinden met items die relevant zijn voor hen. Een goed aanbevelingssysteem voor boeken bijvoorbeeld, beveelt me typisch drie boeken aan die ik met veel plezier zou lezen, die ik nog niet kende, die voldoende van elkaar verschillen en die geschikt zijn om tijdens mijn vakantie van volgende week te lezen. Het bestuderen van aanbevelingssystemen in het bijzonder, en de verbindingen tussen individuen en items in het algemeen, is het onderwerp van *recommendation* onderzoek. Bovendien reikt de relevantie van recommendation onderzoek verder dan het verbinden van individuen met items. Aanbevelingssystemen kunnen bijvoorbeeld, ook genen verbinden met ziektes, biologische doelen met actieve stoffen uit geneesmiddelen, woorden met documenten, onderschriften met foto's, etc.

Collaborative Filtering

Collaborative filtering is één van de belangrijkste problemen in recommendation onderzoek. In de meest abstracte zin is collaborative filtering het probleem waarin de onbestaande zijden in een bipartiete graaf moeten gewogen worden op basis van de bestaande zijden.

Het voorspellen van beoordelingen is de concrete versie van dit probleem die tot voor kort het meeste aandacht kreeg. In deze versie vertegenwoordigt één verzameling van punten in de bipartiete graaf de individuen, en de andere verzameling van punten de items. Verder duidt een zijde met gewicht r tussen individu u en item i aan dat u een beoordeling r gaf aan i. Het doel is de ontbrekende beoordelin-

gen te voorspellen. Om wille van verschillende redenen echter, is de laatste tijd de aandacht voor het voorspellen van beoordelingen gedaald. Ten eerste is het redelijk duur om beoordelingen te verzamelen, in die zin dat het een niet verwaarloosbare inspanning van gebruikers vergt. Ten tweede correleren beoordelingen niet zo goed met het gedrag van gebruikers als men zou verwachten. Gebruikers zijn geneigd hoge beoordelingen te geven aan items waarvan ze denken dat ze die zouden moeten consumeren, zoals bijvoorbeeld een beroemd boek van Dostojewski. In de realiteit lezen dezelfde gebruikers eerder Suske en Wiske strips, hoewel ze die een lagere beoordeling geven. Ten slotte is voor vele toepassingen het voorspellen van beoordelingen geen einddoel, maar slechts een manier om de meest relevante aanbevelingen voor een gebruiker te vinden. Bijgevolg dienen hoge beoordelingen zo accuraat mogelijk te zijn, terwijl lage beoordelingen niet exact hoeven te zijn. Echter, bij het voorspellen van beoordelingen zijn hoge en lage beoordelingen even belangrijk.

Vandaag de dag verschuift de aandacht meer en meer in de richting van collaborative filtering met binaire, positieve data Dit is het onderwerp van dit proefschrift. In deze versie zijn de zijden van de bipartiete graaf ongewogen, en duidt een zijde tussen individu u en item i aan dat u positieve feedback gaf over i. Het doel is in dit geval elke ontbrekende zijde een score toe te kennen die aangeeft in welke mate i een goede aanbeveling is voor u. Binaire, positieve data wordt typisch geassocieerd met impliciete feedback zoals aangekochte producten, bekeken films, geluisterde liedjes, ontleende boeken, geklikte advertenties, etc. Maar het kan ook perfect het resultaat zijn van expliciete feedback, zoals *likes* op sociale netwerk sites.

Naast de bipartiete graaf kunnen er nog vijf soorten van extra informatie beschikbaar zijn. Ten eerste kan er inhoud of meta data van de items beschikbaar zijn. In het geval van boeken bijvoorbeeld, is de inhoud gelijk aan de volledige tekst van het boek en kan de meta data informatie bevatten zoals schrijver, uitgever, jaar van uitgifte etc. Methodes die uitsluitend dit soort informatie gebruiken worden typisch geklasseerd als *inhoud gebaseerd*. Methodes die dit soort informatie combineren met collaborative filtering methodes worden typisch geklasseerd als *hybride*. Ten tweede kan er meta data beschikbaar zijn over de gebruikers, zoals geslacht, leeftijd, woonplaats, etc. Ten derde kunnen gebruikers met elkaar verbonden zijn in een extra unipartiete graaf. Een typisch voorbeeld is een sociaal netwerk tussen gebruikers. Ten vierde, kan een analoog netwerk bestaan tussen items. Ten slotte kan er contextuele informatie beschikbaar zijn zoals locatie, tijd, intentie, gezelschap, toestel, etc. Het aanwenden van deze extra informatie valt buiten het bereik van dit proefschrift.

Verband met andere domeinen

Om het unieke karakter van collaborative filtering te benadrukken, bespreken we de gemeenschappelijkheden en verschillen met twee gerelateerde problemen uit data wetenschappen: classificatie en het ontdekken van associatie-regels.

Ten eerste is collaborative filtering equivalent aan het gezamenlijk oplossen van vele één-klasse classificatie problemen, waarbij elk één-klasse classificatie probleem overeenkomt met één van de items. In het classificatie probleem dat overeenkomt met item i, doet i dienst als klasse en doen alle andere items dienst als features. De

gebruikers die i als gekende voorkeur hebben fungeren als gelabelde voorbeelden, terwijl de andere gebruikers dienst doen als niet gelabelde voorbeelden. Amazon.com heeft bijvoorbeeld meer dan 200 miljoen items in zijn catalogus. Bijgevolg is het oplossen van het collaborative filtering probleem voor Amazon.com equivalent aan het oplossen van 200 miljoen één-klasse classificatie problemen, hetgeen duidelijk een specifieke aanpak vergt. Collaborative filtering is echter meer dan het efficiënt oplossen van vele één-klasse classificatie problemen. Omdat ze sterk verbonden zijn, laat het *gezamenlijk* oplossen van de vele één-klasse classificatie problemen toe dat er informatie gedeeld wordt tussen de verschillende problemen. De verschillende één-klasse classificatie problemen delen het grootste deel van hun features. Terwijl i dienst doet als klasse in één van de vele classificatie problemen, doet het eveneens dienst als feature in alle andere classificatie problemen.

Ten tweede veronderstelt ook het ontdekken van associatie-regels bipartiete, ongewogen data. Daardoor kan het toegepast worden op dezelfde data als diegene die gebruikt wordt voor collaborative filtering. Bovendien kan het aanbevelen van item i aan gebruiker u beschouwd worden als het toepassen van de associatie regel $I(u) \rightarrow i$, waarin $I(u)$ de verzameling is van alle gekende voorkeuren van u. Het doel van associatie-regels ontdekken verschilt echter van dat van collaborative filtering. Als een regel $I(u) \rightarrow i$ cruciaal is om i correct aan u aan te bevelen, maar irrelevant voor de overige data, dan is een hoge score voor de regel gewenst in het geval van collaborative filtering, maar is ze typisch ongewenst voor het ontdekken van associatie-regels.

Bijdragen

Dit proefschrift bevat volgende bijdragen:

- In **Hoofdstuk 2** geven we een overzicht van de state of the art op het vlak van collaborative filtering met binaire, positieve data. De kapstok waaraan dit overzicht is opgehangen, is een innovatief, eengemaakt matrix factorizatie perspectief op collaborative filtering methodes, ook diegene die typisch niet met matrix factorizatie geassocieerd worden, zoals methodes gebaseerd op nearest neighbors of associatie-regels. Vanuit dit perspectief bestaat een collaborative filtering methode uit drie bouwblokken: een matrix factorizatie model, een afwijkingsfunctie en een algoritme voor numerieke minimalisatie. Door de verschillende methodes te vergelijken volgens deze drie dimensies, kunnen we verassende gelijkenissen en cruciale verschillen blootleggen.

- In **Hoofdstuk 3** introduceren we een herformulering die user- en item-based nearest neighbors methodes unificeert. We gebruiken deze herformulering om een nieuwe methode voor te stellen die het beste van beide werelden combineert en beter presteert dan state-of-the-art methodes. Bijkomend stellen we een methode voor om de aanbevelingen gemaakt door onze nieuwe methode op een natuurlijke wijze te verklaren. Verder tonen we aan dat deze methode ook toepasbaar is op user-based nearest neighbors methodes, waarvan gedacht werd dat ze geen natuurlijke verklaring hebben.

- In **Hoofdstuk 4** gaan we dieper in op de situatie waarin meerdere gebruikers dezelfde gebruikersnaam hanteren. Een typische voorbeeld is een gebruik-

ersnaam voor een online winkel die door alle leden van een gezin gedeeld wordt. Traditionele aanbevelingssystemen maken fouten in deze situatie. Indien informatie beschikbaar is over de context waarin transacties gebeuren, zijn context bewuste aanbevelingssystemen de voor de hand liggende oplossing. Vaak is er echter geen informatie beschikbaar over de context waarin transacties plaatsvinden. Daarom introduceren we de uitdaging waarin aanbevelingen moeten gegeven worden aan gedeelde gebruikersnamen zonder te beschikken over contextuele informatie. We stellen ook een oplossing voor, voor alle gevallen waarin het referentie aanbevelingssysteem een item-based collaborative filtering systeem is. Ten slotte illustreren we met experimenten op meerdere datasets de voordelen van onze methode voor het oplossen van de problemen die ontstaan wanneer gebruikersnamen gedeeld worden.

- **Hoofdstuk 5** gaat over het efficiënt berekenen van de neighborhoods voor neighborhood-gebaseerde collaborative filtering methodes. Meer algemeen beschouwen we een grote verzameling van spaarse vectoren in een hoog dimensionale ruimte en onderzoeken we de efficiënte berekening van de k meest gelijkaardige vectoren voor elke vector. Op dit moment bestaan er exacte, efficiënte methodes voor laag dimensionale data. Voor hoog dimensionale data daarentegen, zijn tot hiertoe enkel benaderende, efficiënte methodes voorgesteld. De bestaande aanpakken maken echter geen onderscheid tussen dense en spaarse data. Door te focussen op spaarse data, zijn we in staat om twee methodes voor te stellen, gebaseerd op inverted indexes, die in staat zijn de spaarsheid van de data uit te buiten om efficiënter de exacte k-nearest neighbors graaf te berekenen voor hoog dimensionale, spaarse data.

- **Hoofdstuk 6** sluit af met onze visie op toekomstig onderzoek naar collaborative filtering.

Bibliography

[1] F. Aiolli. Efficient top-n recommendation for very large scale binary rated datasets. In *Proceedings of the 7th ACM Conference on Recommender Systems*, pages 273–280. ACM, 2013.

[2] F. Aiolli. Convex auc optimization for top-n recommendation with implicit feedback. In *Proceedings of the 8th ACM Conference on Recommender Systems*, pages 293–296. ACM, 2014.

[3] S. S. Anand and B. Mobasher. Contextual recommendation. Springer, 2007.

[4] D. C. Anastasiu and G. Karypis. L2ap: Fast cosine similarity search with prefix l-2 norm bounds. In *Proceedings of the 30th IEEE International Conference on Data engineering*, pages 784–795, 2014.

[5] A. Arasu, V. Ganti, and R. Kaushik. Efficient exact set-similarity joins. In *Proceedings of the 32nd International Conference on Very large data bases*, pages 918–929. VLDB Endowment, 2006.

[6] A. Awekar and N. F. Samatova. Fast matching for all pairs similarity search. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology-Volume 01*, pages 295–300. IEEE Computer Society, 2009.

[7] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *Proceedings of the 16th International Conference on World Wide Web*, pages 131–140. ACM, 2007.

[8] A. Bellogin, J. Wang, and P. Castells. Text retrieval methods for item ranking in collaborative filtering. In P. Clough, C. Foley, C. Gurrin, G. Jones, W. Kraaij, H. Lee, and V. Mudoch, editors, *Advances in Information Retrieval*, volume 6611, pages 301–306. Springer Berlin Heidelberg, 2011.

[9] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[10] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere. The million song dataset. In *Proceedings of the 12th International Society for Music Information Retrieval Conference*, pages 591–596. University of Miami, 2011.

[11] C. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, NY, 2006.

[12] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *The Journal of machine Learning research*, 3:993–1022, 2003.

[13] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *Proceedings of the 22nd International Conference on Data engineering*, pages 5–5. IEEE, 2006.

[14] J. Chen, H.-r. Fang, and Y. Saad. Fast approximate k-nn graph construction for high dimensional data via recursive lanczos bisection. *The Journal of Machine Learning Research*, 10:1989–2012, 2009.

[15] E. Christakopoulou and G. Karypis. Hoslim: Higher-order sparse linear method for top-n recommender systems. In *Advances in Knowledge Discovery and Data Mining*, pages 38–49. Springer, 2014.

[16] M. Connor and P. Kumar. Fast construction of k-nearest neighbor graphs for point clouds. *IEEE Transactions on Visualization and Computer Graphics*, 16(4):599–608, 2010.

[17] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM Conference on Recommender Systems*, pages 39–46. ACM, 2010.

[18] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[19] M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems (TOIS)*, 22(1):143–177, 2004.

[20] C. Desrosiers and G. Karypis. A comprehensive survey of neighborhood-based recommendation methods. In *Recommender systems handbook*, pages 107–144. Springer, 2011.

[21] C. Dhanjal, R. Gaudel, and S. Clémençon. Collaborative filtering with localised ranking. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, 2015.

[22] S. Ding and T. Suel. Faster top-k document retrieval using block-max indexes. In *Proceedings of the 34th International ACM SIGIR Conference on Research and development in Information Retrieval*, pages 993–1002, 2011.

[23] W. Dong, C. Moses, and K. Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th International Conference on World wide web*, pages 577–586, 2011.

[24] C. Eckart and G. Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[25] J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010.

[26] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, 1977.

[27] K. Fukunaga and P. M. Narendra. A branch and bound algorithm for computing k-nearest neighbors. *IEEE Transactions on Computers*, 100(7):750–753, 1975.

[28] Z. Gantner, S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. Mymedialite: A free recommender system library. In *Proceedings of the fifth ACM Conference on Recommender Systems*, pages 305–308, 2011.

[29] P. Gopalan, J. Hofman, and D. Blei. Scalable recommendation with hierarchical poisson factorization. In *Proceedings of the Thirti-first Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-15). AUAI Press*, 2015.

[30] P. Gopalan, F. J. Ruiz, R. Ranganath, and D. M. Blei. Bayesian nonparametric poisson factorization for recommendation systems. *Artificial Intelligence and Statistics (AISTATS)*, 33:275–283, 2014.

[31] M. Gori, A. Pucci, V. Roma, and I. Siena. Itemrank: A random-walk based scoring algorithm for recommender engines. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, volume 7, pages 2766–2771, 2007.

[32] Grouplens. ml-1m.zip. http://grouplens.org/datasets/movielens/.

[33] K. Hajebi, Y. Abbasi-Yadkori, H. Shahbazi, and H. Zhang. Fast approximate nearest-neighbor search with k-nearest neighbor graph. In *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 22, page 1312, 2011.

[34] J. Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[35] J. L. Herlocker, J. A. Konstan, and J. Riedl. Explaining collaborative filtering recommendations. In *Proceedings of the ACM conference on Computer Supported Cooperative Work*, pages 241–250. ACM, 2000.

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

[37] T. Hofmann and J. Puzicha. Latent class models for collaborative filtering. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 688–693, 1999.

[38] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the Eighth IEEE International Conference on Data Mining*, pages 263–272. IEEE, 2008.

[39] Z. Huang, X. Li, and H. Chen. Link prediction approach to collaborative filtering. In *Proceedings of the 5th ACM/IEEE-CS joint Conference on Digital libraries*, pages 141–142. ACM, 2005.

[40] Y. Hwang, B. Han, and H.-K. Ahn. A fast nearest neighbor search algorithm by nonlinear embedding. In *Proceedings of the IEEE Conference on Computer vision and pattern recognition*, pages 3053–3060. IEEE, 2012.

[41] D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich. *Recommender systems: an introduction*. Cambridge University Press, 2010.

[42] C. Johnson. Logistic matrix factorization for implicit feedback data. In *Workshop on Distributed Machine Learning and Matrix Computations at the Twenty-eighth Annual Conference on Neural Information Processing Systems (NIPS)*, 2014.

[43] S. Jonassen and S. E. Bratsberg. Efficient compressed inverted index skipping for disjunctive text-queries. In *Advances in Information Retrieval*, pages 530–542. Springer, 2011.

[44] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

[45] S. Kabbur and G. Karypis. Nlmf: Nonlinear matrix factorization methods for top-n recommender systems. In *Workshop Proceedings of the IEEE International Conference on Data Mining*, pages 167–174. IEEE, 2014.

[46] S. Kabbur, X. Ning, and G. Karypis. Fism: factored item similarity models for top-n recommender systems. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge discovery and data mining*, pages 659–667. ACM, 2013.

[47] N. Koenigstein, N. Nice, U. Paquet, and N. Schleyen. The xbox recommender system. In *Proceedings of the sixth ACM Conference on Recommender Systems*, pages 281–284. ACM, 2012.

[48] Y. Koren and R. Bell. Advances in collaborative filtering. In *Recommender systems handbook*, pages 145–186. Springer, 2011.

[49] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.

[50] D. Lee, J. Park, J. Shim, and S.-g. Lee. An efficient similarity join algorithm with cosine similarity predicate. In *Database and Expert Systems Applications*, pages 422–436. Springer, 2010.

[51] B. Leibe, K. Mikolajczyk, and B. Schiele. Efficient clustering and matching for object class recognition. In *BMVC*, pages 789–798, 2006.

[52] D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031, 2007.

[53] W. Lin, S. A. Alvarez, and C. Ruiz. Efficient adaptive-support association rule mining for recommender systems. *Data mining and knowledge discovery*, 6(1):83–105, 2002.

[54] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[55] J. Masthoff. Group recommender systems: Combining individual models. In P. B. Kantor, L. Rokach, F. Ricci, and B. Shapira, editors, *Recommender Systems Handbook*, pages 677–702. Springer, 2011.

[56] G. V. Menezes, J. M. Almeida, F. Belém, M. A. Gonçalves, A. Lacerda, E. S. De Moura, G. L. Pappa, A. Veloso, and N. Ziviani. Demand-driven tag recommendation. In *Machine Learning and Knowledge Discovery in Databases*, pages 402–417. Springer, 2010.

[57] A. Mnih and R. Salakhutdinov. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264, 2007.

[58] B. Mobasher, H. Dai, T. Luo, and M. Nakagawa. Effective personalization based on association rule discovery from web usage data. In *Proceedings of the 3rd International workshop on Web information and data management*, pages 9–15. ACM, 2001.

[59] M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, 2014.

[60] X. Ning and G. Karypis. Slim: Sparse linear methods for top-n recommender systems. In *Proceedings of the 11th IEEE International Conference on Data Mining*, pages 497–506. IEEE, 2011.

[61] NLTK-Project. Natural language toolkit. http://www.nltk.org/.

[62] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: bringing order to the web. 1999.

[63] C. Palmisano, A. Tuzhilin, and M. Gorgolione. Using context to improve predictive modeling of customers in personalization applications. 20(11):1535–1549, 2008.

[64] R. Pan and M. Scholz. Mind the gaps: weighting the unknown in large-scale one-class collaborative filtering. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge discovery and data mining*, pages 667–676. ACM, 2009.

[65] R. Pan, Y. Zhou, B. Cao, N. N. Liu, R. Lukose, M. Scholz, and Q. Yang. One-class collaborative filtering. In *Proceedings of the Eighth IEEE International Conference on Data Mining*, pages 502–511. IEEE, 2008.

[66] W. Pan and L. Chen. Cofiset: Collaborative filtering via learning pairwise preferences over item-sets. In *Proceedings of the 13th SIAM International Conference on Data Mining*, pages 180–188, 2013.

[67] W. Pan and L. Chen. Gbpr: Group preference based bayesian personalized ranking for one-class collaborative filtering. In *Proceedings of the Twenty-Third International joint Conference on Artificial Intelligence*, pages 2691–2697. AAAI Press, 2013.

[68] U. Paquet and N. Koenigstein. One-class collaborative filtering with random graphs. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 999–1008, 2013.

[69] R. Paredes, E. Chávez, K. Figueroa, and G. Navarro. Practical construction of k-nearest neighbor graphs in metric spaces. In *Experimental Algorithms*, pages 85–97. Springer, 2006.

[70] Y. Park, S. Park, S.-g. Lee, and W. Jung. Scalable k-nearest neighbor graph construction based on greedy filtering. In *Proceedings of the 22nd International Conference on World Wide Web companion*, pages 227–228, 2013.

[71] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD cup and workshop*, volume 2007, pages 5–8, 2007.

[72] I. Pilászy, D. Zibriczky, and D. Tikk. Fast als-based matrix factorization for explicit and implicit feedback datasets. In *Proceedings of the fourth ACM Conference on Recommender Systems*, pages 71–78. ACM, 2010.

[73] B. Pradel, N. Usunier, and P. Gallinari. Ranking with non-random missing ratings: Influence of popularity and positivity on evaluation metrics. In *Proceedings of the sixth ACM Conference on Recommender Systems*, pages 147–154, 2012.

[74] S. Rendle and C. Freudenthaler. Improving pairwise learning for item recommendation from implicit feedback. In *Proceedings of the 7th ACM International Conference on Web search and data mining*, pages 273–282. ACM, 2014.

[75] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 452–461. AUAI Press, 2009.

[76] J. D. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the 22nd International Conference on Machine learning*, pages 713–719. ACM, 2005.

[77] F. Ricci, L. Rokach, B. Shapira, and P. Kantor. *Recommender Systems Handbook*. Springer, Boston, MA, 2011.

[78] S. Sarawagi and A. Kirpal. Efficient set joins on similarity predicates. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of data*, pages 743–754, 2004.

[79] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Analysis of recommendation algorithms for e-commerce. In *Proceedings of the 2nd ACM Conference on Electronic commerce*, pages 158–167. ACM, 2000.

[80] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Application of dimensionality reduction in recommender system-a case study. Technical report, DTIC Document, 2000.

[81] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, pages 285–295. ACM, 2001.

[82] Y. Shi, A. Karatzoglou, L. Baltrunas, M. Larson, A. Hanjalic, and N. Oliver. Tfmap: Optimizing map for top-n context-aware recommendation. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 155–164. ACM, 2012.

[83] Y. Shi, A. Karatzoglou, L. Baltrunas, M. Larson, N. Oliver, and A. Hanjalic. Climf: learning to maximize reciprocal rank with collaborative less-is-more filtering. In *Proceedings of the sixth ACM Conference on Recommender Systems*, pages 139–146. ACM, 2012.

[84] B. Sigurbjörnsson and R. Van Zwol. Flickr tag recommendation based on collective knowledge. In *The 17th International Conference on World Wide Web*, pages 327–336. ACM, 2008.

[85] V. Sindhwani, S. S. Bucak, J. Hu, and A. Mojsilovic. One-class matrix completion with low-density factorizations. In *Proceedings of the 10th IEEE International Conference on Data Mining*, pages 1055–1060. IEEE, 2010.

[86] R. F. Sproull. Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica*, 6:579–589, 1991.

[87] H. Steck. Training and testing of recommender systems on data missing not at random. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge discovery and data mining*, pages 713–722. ACM, 2010.

[88] H. Steck. Item popularity and recommendation accuracy. In *Proceedings of the fifth ACM Conference on Recommender Systems*, pages 125–132. ACM, 2011.

[89] H. Steck. Gaussian ranking by matrix factorization. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 115–122. ACM, 2015.

[90] T. Strohman and W. B. Croft. Efficient document retrieval in main memory. In *Proceedings of the 30th annual International ACM SIGIR Conference on Research and development in information retrieval*, pages 175–182, 2007.

[91] B. Stroustrup. Software development for infrastructure. *IEEE Computer*, 45(1):47–58, 2012.

[92] P. Symeonidis, A. Nanopoulos, A. N. Papadopoulos, and Y. Manolopoulos. Nearest-biclusters collaborative filtering based on constant and coherent values. *Information retrieval*, 11(1):51–75, 2008.

[93] G. Takács and D. Tikk. Alternating least squares for personalized ranking. In *Proceedings of the sixth ACM Conference on Recommender Systems*, pages 83–90. ACM, 2012.

[94] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su. Arnetminer: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge discovery and data mining*, pages 990–998. ACM, 2008.

[95] N. Tintarev. Explenations of recommendations. In *Proceedings of the first ACM Conference on Recommender Systems*, pages 203–206, 2007.

[96] A. Töscher and M. Jahrer. Collaborative filtering ensemble for ranking. *Journal of Machine Learning Research W&CP: Proceedings of KDD Cup 2011*, 18:61–74, 2012.

[97] H. Turtle and J. Flood. Query evaluation: strategies and optimizations. *Information Processing & Management*, 31(6):831–850, 1995.

[98] Twitter. Twitter streaming apis. https://dev.twitter.com/streaming/overview.

[99] L. H. Ungar and D. P. Foster. Clustering methods for collaborative filtering. In *AAAI workshop on recommendation systems*, volume 1, pages 114–129, 1998.

[100] M. van Leeuwen and D. Puspitaningrum. Improving tag recommendation using few associations. In *Advances in Intelligent Data Analysis XI*, pages 184–194. Springer, 2012.

[101] K. Verstrepen, K. Bhaduri, and B. Goethals. A survey of collaborative filtering methods for binary, positive-only data. In *Under submission*, 2015.

[102] K. Verstrepen and B. Goethals. Unifying nearest neighbors collaborative filtering. In *Proceedings of the 8th ACM Conference on Recommender Systems*, pages 177–184. ACM, 2014.

[103] K. Verstrepen and B. Goethals. Efficiently computing the exact k-nn graph for high dimensional sparse data. In *Under submission*, 2015.

[104] K. Verstrepen and B. Goethals. Top-n recommendation for shared accounts. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 59–66. ACM, 2015.

[105] J. Wang, A. P. de Vries, and M. J. Reinders. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *Proceedings of the 29th International ACM SIGIR Conference on Research and development in information retrieval*, pages 501–508, 2006.

[106] J. Wang, A. P. De Vries, and M. J. Reinders. A user-item relevance model for log-based collaborative filtering. In *Advances in Information Retrieval*, pages 37–48. Springer, 2006.

[107] X. Wang. A fast exact k-nearest neighbors algorithm for high dimensional search using k-means clustering and triangle inequality. In *Proceedings of the International Joint Conference on Neural networks*, pages 1293–1299. IEEE, 2011.

[108] J. Weston, S. Bengio, and N. Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *Proceedings of the 22th Interantional Joint Conference on Artifical Intelligence*, volume 11, pages 2764–2770, 2011.

[109] J. Weston, R. J. Weiss, and H. Yee. Nonlinear latent factorization by embedding multiple user interests. In *Proceedings of the 7th ACM Conference on Recommender Systems*, pages 65–68. ACM, 2013.

[110] J. Weston, H. Yee, and R. J. Weiss. Learning to rank recommendations with the k-order statistic loss. In *Proceedings of the 7th ACM Conference on Recommender Systems*, pages 245–248. ACM, 2013.

[111] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, 1999.

[112] C. Xiao, W. Wang, X. Lin, and H. Shang. Top-k set similarity joins. In *Proceedings of the 25th IEEE International Conference on Data engineering*, pages 916–927. IEEE, 2009.

[113] Yahoo!Research. Yahoo_webscope_r3.tgz. `http://research.yahoo.com/Academic_Relations`.

[114] Y. Yao, H. Tong, G. Yan, F. Xu, X. Zhang, B. K. Szymanski, and J. Lu. Dual-regularized one-class collaborative filtering. In *Proceedings of the 23rd ACM International Conference on Information and Knowledge Management*, pages 759–768. ACM, 2014.

[115] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA*, volume 93, pages 311–321, 1993.

[116] C. Yu, B. Cui, S. Wang, and J. Su. Efficient index-based knn join processing for high-dimensional data. *Information and Software Technology*, 49(4):332–344, 2007.

[117] C. Yu, L. V. Lakshmanan, and S. Amer-Yahia. Recommendation diversification using explanations. In *Proceedings of the 25th IEEE International Conference on Data Engineering*, pages 1299–1302. IEEE, 2009.

[118] A. Zhang, N. Fawaz, S. Ioannidis, and A. Montanari. Guess who rated this movie: Identifying users through subspace clustering. In *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence*, pages 944–953, 2012.

[119] W. Zhang, T. Chen, J. Wang, and Y. Yu. Optimizing top-n collaborative filtering via dynamic negative item sampling. In *Proceedings of the 36th International ACM SIGIR Conference on Research and development in information retrieval*, pages 785–788. ACM, 2013.

[120] H. Zhong, W. Pan, C. Xu, Z. Yin, and Z. Ming. Adaptive pairwise preference learning for collaborative recommendation with implicit feedbacks. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 1999–2002. ACM, 2014.

[121] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan. Large-scale parallel collaborative filtering for the netflix prize. In *Algorithmic Aspects in Information and Management*, pages 337–348. Springer, 2008.

[122] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th International Conference on World Wide Web*, pages 22–32. ACM, 2005.

[123] A. Zubiaga. Enhancing naviagion on wikipedia with social tags. arXiv:1202.5469, 2012.