Bachelor's Thesis

A Hyper-Parameter-Tuned, Confidence-Weighted Collaborative Filtering Model for Implicit Feedback

Department of Statistics
Ludwig-Maximilians-Universität München

By Lisa Wimmer
Under the supervision of Prof. Dr. Christian Heumann

Submitted in partial fulfillment of the requirements for the degree of
Bachelor of Science
Munich, October 24th, 2019

Abstract

Addressing the complexity of choice in an increasingly labyrinthine consumption environment, it has become common for providers of products and services to offer recommendations to their users as decision support. These recommendations are obtained through recommender systems, the most widely applied of which are collaborative filtering models. Collaborative filtering exploits users' feedback on items. However, in many practical settings, only implicit feedback in the form of binary information about the presence or absence of an action - for instance, a purchase or a click - is available. The inherent difficulty with implicit feedback is that lack of preference cannot be concluded from the absence of an action. The unobserved action might indeed reflect disapproval, or be attributed to the fact that the user is not even aware of the item in question. Feedback thus exists only in positive or missing form. Latent factor models have been known to handle this type of data input reasonably well, compressing the high-dimensional information into a low-dimensional factor space in which both users and items are represented. In 2008, Yifan Hu, Yehuda Koren and Chris Volinsky proposed a collaborative filtering model which copes with the ambiguity of implicit feedback through assigning confidence-based weights to observations. This Thesis implements the postulated model for retail company data, putting special emphasis on hyper-parameter optimization carried out via grid and random search. While both methods yield satisfying results, random search does so more efficiently and is thus found to be preferable. Furthermore, the results indicate that quality aspects beyond prediction accuracy should be explicitly accounted for.

Contents

List of Figures

List of Tables

1 Introduction

The world of consumption has evolved into a complex grid of connections between individuals and points of interest which increasingly form irrespective of common geographical occurrence. In order to keep the concepts described in the following applicable to various forms of consumption (which might be purchasing a tangible product or listening to a song online), the individuals consuming will be referred to as *users* while objects of consumption will be denoted as *items*. Users enjoy a nearly infinite choice of items and providers, spurred most notably through the rise of the internet. While in the past firms may have successfully distinguished themselves from competitors by establishing brand awareness or by merely occupying a certain area as monopolists, phenomena such as search engines and online shopping have changed the field of competition fundamentally. This tilt of the market towards a vast supply side has prompted providers to apply measures that render their items more relevant to their clientele (Häubl and Trifts, 2000). Users will ultimately regard an item as relevant to them, and thus consume it, if it fulfills their needs. For many items though, consumption is not necessarily preceded by feeling a distinctive need. Rather, a hitherto latent or even non-existing desire surfaces through being affected by marketing activities. Suppliers make use of this possibility of creating needs by offering *recommendations* to the user. Such recommendations come in many ways, from movie tips on streaming websites to proposals for additional purchases at an online shop checkout (Schafer et al., 1999).

Precondition to making meaningful recommendations, in a sense that they result in consumption, is a profound knowledge of the user's interests. To this end several forms of *recommender systems* have been developed (Melville and Sindhwani, 2017). One approach that has gained wide popularity is so-called *collaborative filtering*, which exploits users' feedback towards items on offer. In its original form, collaborative filtering was designed to process feedback in the form of ratings on a fixed scale that clearly indicate whether the user does or does not approve of the respective item (see for example Su and Koshgoftaar, 2009). However, such rating data is only available to limited extent as many suppliers do not operate systems to gather explicit ratings on their items. Instead, recent work has focused on binary feedback data derived from the information whether or not a user has performed a certain action, typically a purchase, on an item (*implicit feedback*). Such data exists in abundance but lacks a central feature of rating data: the presence of negative feedback, that is, when the action under observation has not been performed this does not necessarily reflect disapproval of the item. The user may simply not be aware of the item or be hindered from consumption in some other way (Pan et al., 2008). In their oft-cited work Yifan Hu, Yehuda Koren and Chris Volinsky addressed this problem by assigning different levels of confidence to observed and unobserved actions (Hu et al., 2008). Their model will henceforth be referred to as the *Hu-Koren-Volinsky model*. While several replications of their experimental studies have corroborated the success of this approach, the quality of recommendations greatly depends on the value of model hyper-parameters. This Thesis will strive to implement the algorithm proposed by Hu et al. (2008) for a set of retail data and focus on optimizing the outcome by tuning the hyper-parameters to the best possible fit. All analytical work is carried out using the software R (R Core Team, 2019).

The Thesis starts by explaining the functionality of collaborative-filtering-based recommender systems with special regard to implicit feedback and proceeds to describing latent factor models as the specific model class of the Hu-Koren-Volinsky model. Section 4 explains the process of hyper-parameter tuning, contrasting two selected approaches that will later serve as tuning methods. In Section 3, the Hu-Koren-Volinsky model is presented in detail. The model is then fitted on real data in Section 5, its hyper-parameters optimized with the two different methods outlined in Section 4. Eventually, the concluding chapter reflects on the results obtained throughout the Thesis and gives an outlook on aspects which remain open.

2 Collaborative-Filtering-Based Recommender Systems

2.1 Recommender Systems

In research on the topic various definitions of the term *recommender system*, each with different emphasis, have been developed. This Thesis will adopt the key elements of the proposal by Bobadilla et al. (2013), which is considered to incorporate all important aspects: recommender systems gather information on user preferences with respect to a set of items (observed interactions) in order to predict preferences of these users for previously not consumed items (unobserved interactions), thereby enabling the operator of the system to offer recommendations to the user (Bobadilla et al., 2013). Preferences can basically be estimated in two different ways. Some models explicitly predict ratings of users on all items, although the term *rating* should be interpreted simply as a quantified expression of preference which does not necessarily involve assigning scores on a fixed scale. Section 2.2.2 provides more detail on the types of ratings recommender systems utilize. Other models content themselves with predicting a ranking list of top-N relevant items. As the latter case applies to the Hu-Koren-Volinsky model examined in this Thesis, the task of prediction will be regarded equivalent to producing a ranking of items rather than explicit estimation of ratings. Consequently, a *recommendation* shall be constituted by suggesting to the user an ordered list of N items the user is most likely to rate highly (Aggarwal, 2016, Chapter 1.2). It is important to note that recommendations in this context are understood to be *personalized*, meaning each user receives a different suggestion of items according to their individual profile. Obviously, the generation of such personalized recommendations requires more sophisticated algorithms than displaying the same set of items, for instance determined by item popularity, to all users (Ricci et al., 2011, Chapter 1.3).

The reason that recommender systems have come into existence is the complex choice of potentially suitable items users nowadays face in consumption. This is true especially in the online environment where the offer is not restrained by geographical proximity between supplier and user. Häubl and Trifts (2000) argue that users in such settings undergo a two-stage decision process in selecting items to be consumed. The first step of decision-making typically comprises researching large amounts of available items and narrowing down the choice to a small subset. This pre-selection enables the user to perform more in-depth comparison among the remaining items and eventually decide on which items to consume in the second step.

To prevent users from aborting the decision process because they deem the choice overwhelmingly complex, suppliers seek to assist the user in step one by offering a relatively small, pre-filtered set of recommendations which can then be further scrutinized (Häubl and Trifts, 2000). This way, the natural social process of collecting advice from trusted sources to facilitate decision-making is supported by an automated system (Melville and Sindhwani, 2017).

Naturally, the central objective of a recommender system must be to provide useful recommendations that result in consumption and thus help generate revenue. This can be broken down into more concrete intermediate goals. Ricci et al. (2011, Chapter 1.2) lay out five objectives recommender systems aim at fulfilling:

(a) **Increase conversion**. This goal refers to users accepting the recommendation and actually consuming an item, which is arguably the most important feature of a recommender system. In many applications the intention is provoking the consumer to purchase an item but the stimulation could also be directed towards reading an article or watching a video.

(b) **Increase item diversity**. Personalized recommendations offer a chance for suppliers to promote more diverse items and thus present the whole palette instead of only the most popular items. The latter is often the case in non-personalized marketing activities hoping to appeal to a broad range of users.

(c) **Increase user satisfaction**. Recommender systems strive to improve user experience by assisting the user journey through relevant recommendations. Ideally, the user will find consumption easy and enjoyable.

(d) **Increase loyalty**. Closely linked to user satisfaction, suppliers are interested in enhancing user loyalty. The more interactions between user and recommender system can be used to train the recommendation algorithm, the more accurate and relevant the suggestions become.

(e) **Increase knowledge on users**. In their function of collecting information on user preferences recommender systems serve to build a sound knowledge base on users. Apart from recommendations this can be utilized in further marketing activities.

While all systems share these common objectives, they may differ greatly in design. A brief overview on types of recommender systems shall help to place the later described model into context. Differences predominantly concern data sources used and algorithms implemented. Recommender systems are categorized according to proposals by Sivapalan et al. (2014) and Ricci et al. (2011, Chapter 1.4) which, in combination, are considered to be exhaustive. As this Thesis focuses on collaborative filtering, and latent factor models in particular, though, other approaches are not presented in detail.

(a) **Association rules**. Such rules rely on the common occurrence of items in bundles of consumption that are referred to as transactions or baskets. If subsets A and B of an entire item set S are frequently consumed within one transaction, it can be concluded that a user who has consumed A will be interested in consuming B with a certain probability (Sivapalan et al., 2014).

This sequence is captured by an association rule $A \Rightarrow B$. The association is considered to have *support* s if $s\%$ of transactions contain $A \cup B$ (Cheung et al., 1996). Rules with support exceeding some threshold can then be used for making recommendations. However, association rules require a lot of computation time and are thus not well suited to the large data bases most applications feature. As they do not draw on user-specific information, they may still be helpful for recommendations to new users on whom no further data is available (Sivapalan et al., 2014).

(b) **Content-based recommender systems**. Content-based models filter items similar to those the user has consumed in the past. Similarity is based on item features such as the genre of a movie or the technical qualities of a product (Ricci et al., 2011, Chapter 1.4). The process of finding similar items can be carried out by different algorithms like clustering methods or k-nearest neighbour classification. Major drawbacks of content-based systems include the difficulty of recommending items to new users and the need for clearly structured data with well-defined content features. Also, relying on feature similarity fails to account for the variations in item popularity, so there is no differentiation between high- and low-selling items so long as they share the same features (Sivapalan et al., 2014).

(c) **Knowledge-based recommender systems**. These systems presume domain-specific knowledge about user requirements. Recommendations are designed to maximize utility to users by suggesting solutions to their needs (Ricci et al., 2011, Chapter 1.4). Rather than exploiting vast databases, knowledge-based algorithms feed from information users insert about their requirements through search filters or similar funnelling mechanisms. Recommendations mainly rely on item similarities and retrieval strategies which prioritize similarities with respect to the overall utility to the user. Consequently, in order to produce useful recommendations, a high amount of costly knowledge engineering is necessary. Once sufficient knowledge has been amassed, though, knowledge-based systems can handle new as well as long-time users and dynamically adapt to changing user preferences (Burke, 2000).

(d) **Community-based recommender systems**. Community-based or social recommender systems make explicit use of the above-mentioned social process of relying on advice from trusted sources in decision-making. To this end, information about the social relations between users is taken into account; recommendations are based on how individuals from the user's personal network have rated items. Social media are a natural application for community-based systems (Ricci et al., 2011, Chapter 1.4). For new users without any known consumption history, inserting information about the user's social connections may improve recommendations, although this advantage becomes irrelevant when substantial interactions with other individuals are also absent. Another open issue with community-based systems is the consideration of distrust, which, if not incorporated into the model, may lead to recommendations backfiring (Victor et al., 2011).

(e) **Demographics-based recommender systems**. This type of recommender system utilizes basic demographic information on users to produce recommendations, including gender and native language. Implicitly, users are segmented into demographically similar groups which are expected to share common preferences. While the sophistication of such an approach is clearly limited, in some contexts it may be a low-cost and yet effective solution that does not require information on past consumption behavior nor high computational efforts (Ricci et al., 2011, Chapter 1.4).

(f) **Collaborative filtering**. Lastly, there is a number of methods referred to as collaborative filtering. These use ratings from active users on the item set as input and have been known to achieve satisfying results, which is why they have gained wide popularity (Sivapalan et al., 2014). As the Hu-Koren-Volinsky Model is itself part of the collaborative-filtering-method family, the following section will present a detailed explanation on the topic.

The variety of approaches described above underlines the applicability of recommender systems to a broad range of contexts that mainly differ in the data base used. Besides, hybrid systems have been developed in an attempt to mitigate the drawbacks of single approaches (Ricci et al., 2011, Chapter 1.4). Section 2 will now elaborate on collaborative filtering models with special regard to implicit feedback data.

2.2 Collaborative Filtering

2.2.1 Principles of Collaborative Filtering

The term *collaborative filtering* was coined by the developers of the first actual recommender system called Tapestry (Hu et al., 2008). This algorithm was designed in the early 1990s to help users filter relevant content out of electronic mail which came at ever larger volume. While Tapestry fed on user annotations and thus contained elements of a content-based system, its developers also sought to pool feedback from various users in order to facilitate filtering, which they called a *collaborative* approach (Goldberg et al., 1992). Meanwhile, collaborative filtering approaches have become ubiquitous in recommendation tasks. Implementations brought forward after Tapestry increasingly automated the recommendation process. Further pioneering systems include GroupLens (news articles), Ringo (music), and BellCore Video Recommender (movies). A surge in popularity was arguably sparked by retail company Amazon's adoption of collaborative filtering as its primary generator of user recommendations. The competition set up by movie streaming provider Netflix in 2006, where a prize was awarded for substantial improvement of the firm's then recommendation algorithm, eventually rendered collaborative filtering a subject of active research and development (Ekstrand et al., 2010).

In a generalized setting, collaborative filtering exploits historical information on observed user-item interactions to predict hitherto unobserved interactions. The basic collaborative filtering mechanism stems from the idea of high correlations between observed ratings across users and items which allow for estimating values in the unobserved cases. As the following section on model types will show, algorithms either focus on inter-item correlations, inter-user correlations, or both simultaneously (Aggarwal, 2016, Chapter 1.3). The classic collaborative filtering set-up[1] thus

[1] Throughout this Thesis, notation for all elements of collaborative filtering shall be kept stringent. Therefore, symbols may at times deviate from those used in the cited sources.

consists of a set of users, U = {user 1, user 2, ..., user m} of size m, and a set of items, I = {item 1, item 2, ..., item n} of size n. Typically m is considerably greater than n and both figures are large. For each user u there is a subset of items, $I_u \subseteq I$, that the user has provided ratings for. Note that I_u may be a null-set if the user has not yet consumed any of the items of interest. The interactions between users and items are represented in a user-item ratings matrix $R \in \mathbb{R}^{m \times n}$, each of whose elements r_{ui} represents the rating r_{ui} of user u on item i (Sarwar et al., 2001). In relying solely on these user-item interactions, collaborative filtering systems are domain-free and do not require any further information such as item properties or user profiles. Rather, they capture underlying and often elusive aspects of the data by exploring correlation structures to be found across the ratings matrix, since data of users with similar tastes or items with similar qualities are assumed to be strongly correlated (Hu et al., 2008). Before describing how this task can be approached, however, it must be clarified which types of data are encountered in collaborative filtering settings and how these affect the process of filtering items to recommend.

2.2.2 Implicit Feedback

As has just been outlined, collaborative filtering systems draw on user-item interactions displayed as ratings. These ratings, or, more generally, feedback of users on items, occurs in either *explicit* or *implicit* form (see for example Sarwar et al., 2001).

Explicit Feedback

Explicit feedback is often provided through ratings in the narrower sense. Users assign values to items out of some ordinal set which is typically discretized and offers the possibility to express both like and dislike. Widely used examples of such rating scales include sets like $\{1, 2, ..., 5\}$ and {"poor", "mediocre", "excellent"} (Aggarwal, 2016, Chapter 1.3.1.1). Obviously, explicit ratings convey a high amount of information on user-item interactions and are thus valuable input to collaborative filtering. Alas, obtaining explicit feedback proves to be difficult in practice as users are often reluctant to state their opinion, if at all there is a platform for ratings. This might, as Jawaheer et al. (2010) argue, be due to the cognitive effort users frequently are not willing to make. Therefore recent research has focused on an alternative type of feedback that comes in greater abundance.

Implicit Feedback

Implicit feedback can be gathered from various environments where users do not even intend to provide feedback of any kind: information is extracted from user behavior. User activities reflecting their attitude towards items include actual consumption, clicks on item websites and similar expressions of interest. Implicit feedback is inherently positive as it either takes on values of the frequency of the observed activity or none at all. It is this imbalance, precisely, that renders filtering implicit feedback so difficult. Whereas low values in explicit rating environments can be safely assumed to indicate disapproval, missing values in implicit feedback do by no means allow for such inference. Some unobserved values may indeed reflect disapproval, but the absence of interaction can be attributed to various other reasons as well. For online applications in particular, which tend to host immense amounts of items, users must be expected to be simply unaware of a substantial share of items. Consequently, implicit feedback is noisier and less expressive than explicit feedback. Its sheer abundance and still considerable informativeness have nonetheless sustained its popularity (Jawaheer et al., 2010).

Since implicit feedback offers by far greater applicability, it is the focus of this Thesis. In order to illustrate more clearly the nature of the data used, an exemplary ratings matrix with implicit feedback is shown in Figure 1. Positive entries indicate how often user u has interacted with item i, while missing values ($*$) occur where no such interaction has yet taken place. For instance, user 1 has had one interaction with item 2, which in a classic shopping environment could mean user 1 has purchased item 2 once.

$$R = (\, r_{ui}) \;=\; \left(\begin{array}{cccccc} & \text{item } 1 & \text{item } 2 & \text{item } 3 & \cdots & \text{item } n \\ \text{user } 1 & \ast & 1 & \ast & \cdots & \vdots \\ \text{user } 2 & \ast & \ast & 2 & \cdots & \vdots \\ \text{user } 3 & 1 & \ast & 1 & \cdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \text{user } m & \cdots & \cdots & \cdots & \cdots & \ast \end{array}\right)$$

Figure 1: Exemplary ratings matrix $R \in \mathbb{R}^{m \times n}$ for m users and n items. If user u has interacted with item i, r_{ui} corresponds to the number of interactions, else to $*$ (Source: own illustration).

2.2.3 Model-Based Collaborative Filtering

As has been described above, collaborative filtering techniques are based on the assumption that user-item ratings are correlated across users and items. Within collaborative filtering there are various model sub-types which differ rather strongly in their approach to exploiting this correlation structure. Aggarwal (2016, Chapters 2 and 3) provides a useful categorization of models as illustrated in Figure 2. For extensive elaboration on each of the different collaborative filtering techniques see therefore Aggarwal (2016). Here, the collaborative filtering model family shall only be briefly introduced in order to put the Hu-Koren-Volinsky model, which is a *latent factor model*, into context. Methods can be broadly classified into *neighborhood-based* and *model-based* approaches.

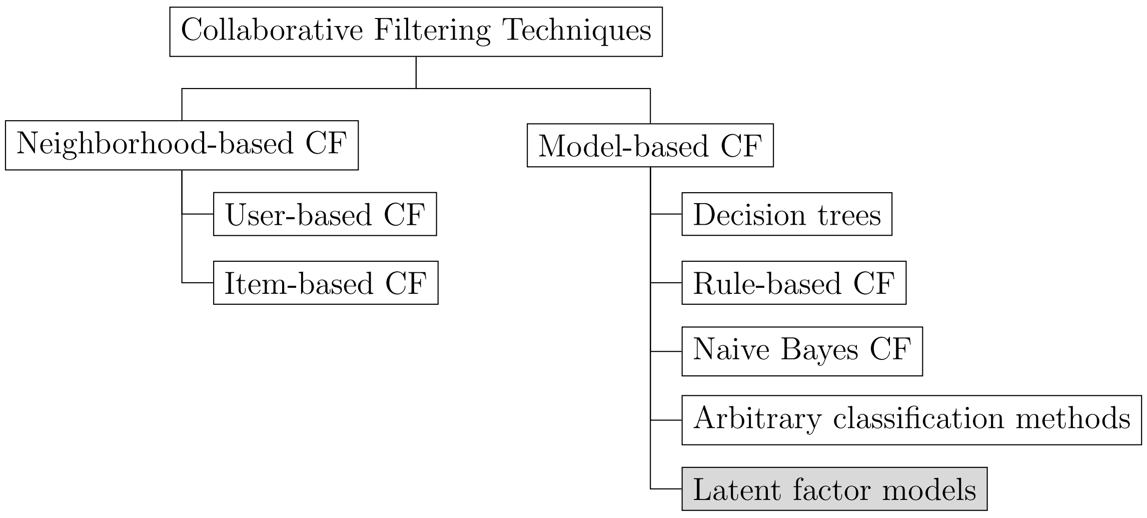


Figure 2: Categorization of collaborative filtering (CF) techniques. Latent factor models on which this Thesis focuses are highlighted in gray. (Source: own illustration after Aggarwal (2016, Chapters 2 and 3)).

Neighborhood-Based Collaborative Filtering

Neighborhood-based collaborative filtering is centered around measuring similarity. The underlying idea is that high similarity between the observed entries of two vectors can be assumed to hold true also for the remaining, possibly unobserved entries. Consequently, for elements where one vector has missing values predictions can be based on the corresponding non-missing values of a similar vector (Aggarwal, 2016, Chapter 2.1). Similarity can be quantified in various ways; frequently used measures in the collaborative filtering context include correlation coefficients of vectors and the cosine of the angle in between. The concept of obtaining information from similar vectors can be applied to both the user and the item dimension of the ratings matrix R, leading to *user-based* and *item-based* forms of neighborhood-based filtering. In both cases, which share a complementary relationship, deriving recommendations is the same basic two-step process. The first step consists of computing pairwise similarities between vectors. In the second step, average ratings across these vectors are utilized for prediction. In essence, user-based methods take the average of ratings from those users who are similar to the target user (*neighborhood*); item-based methods form predictions as the target user's average rating on those items that are deemed similar to the item in question (Su and Koshgoftaar, 2009). Neighborhood-based collaborative filtering algorithms are intuitive and easy to implement. However, prediction quality has been shown to deteriorate in environments with sparse data and scalability is limited. In most settings that feature large amounts of data with only few observed interactions, neighborhood-based systems are thus not well applicable (Su and Koshgoftaar, 2009).

Model-Based Collaborative Filtering

Model-based collaborative filtering serves to mitigate these shortcomings to some extent. Similar to neighborhood-based algorithms, these approaches utilize the presumably high correlations across users and items of the ratings matrix. However, rather than relying on pairwise similarities between users or between items, they attempt at detecting more complex patterns that affect both dimensions simultaneously (Su and Koshgoftaar, 2009). The underlying belief is that with such a pattern, the often extremely high-dimensional space user-item interactions create can be decomposed into a much lower-dimensional representation of users and items that still holds a sufficient share of information (Ekstrand et al., 2010). This task is not unique to collaborative filtering problems, which is why some of the methods within the model-based branch are familiar from other contexts. In particular, decision trees, Bayesian approaches and arbitrary classification methods usually solve classification problems. Since classification is essentially a specific case of a matrix completion problem, collaborative filtering with implicit feedback data can also be assessed as a one-class classification problem, so the use of such methods seems natural. In general, model-based algorithms are often preferable to other techniques as their core quality of dimensionality reduction allows for a decrease in computation complexity. A user-based system (which is usually more complex than the item-based alternative since applications tend to have more users than items) requires computation time of $O(m^2)$, m being the number of users[2] (Aggarwal, 2016, Chapter 3.1).

The following section will now take a closer look on latent factor models and lay out how this type of model-based technique solves the matrix completion problem of collaborative filtering. Again, within latent factor models, there are sub-types which use slightly different mathematical methods. Section 3 will explain in full detail how the Hu-Koren-Volinsky model derives recommendations step by step.

2.3 Collaborative Filtering with Latent Factor Models

2.3.1 Concept of Latent Factors

Latent factor models are based on the idea that while the interactions of m users with n different items create a highly complex situation of dimension $m \times n$, they can in fact be expressed by a small amount of factors that characterize both users and items. Representing users and items through individual values of these factors is thus deemed sufficient for explaining their interactions (Koren, 2008). Due to the fact that they are generally not observable these factors are called *latent*. Latent variables play a crucial role in many contexts where the phenomena of interest cannot be observed directly and must therefore be approximated through manifest variables that are thought to be closely correlated to the latent concepts. One classic example that actually marks the origins of factor analysis is psychologist Charles Spearman's examination in which he sought to measure intelligence, clearly a latent phenomenon, via quantifiable performance on different intellectual tests. As Loehlin and Beaujean (2017) put it, Spearman's work must be regarded as a *confirmatory* approach to factor analysis: he had a clear hypothesis of the nature of latent factors beyond intellectual performance. By contrast, in what they call *exploratory* factor analysis, observations of manifest variables are traced back to factors hitherto unknown (Loehlin and Beaujean, 2017, Chapter 1). The latter is the case in the collaborative filtering context. Here, latent factors are often thought of as user affinities, or interests, that are reflected by items to a certain extent. Users

[2] *Big O notation* is a common way of documenting time complexity of algorithms which draws comparisons from input sizes expressed in natural numbers. This form of denoting complexity is arguably an oversimplification (see for example Krone et al. (2003)) but sufficient to give an idea of complexity relations.

with a distinct affinity for some factor are then expected to rate an item highly which incorporates this factor to a great degree (Koren, 2008). More precisely, a user u will interact with item i if both share high positive correlations with the same underlying factors. In factor analysis this is also referred to as high factor *loading* (Loehlin and Beaujean, 2017, Chapter 1). The complex structural relationship between latent and manifest variables is frequently illustrated through path diagrams. Figure 3[3] shows an exemplary path diagram for a collaborative filtering setting with $m = 6$ users and $n = 3$ items where user-item interactions are assumed to be explicable via $k = 2$ latent factors. The latter may be correlated with a correlation coefficient of φ_{12}. x_{uf} and y_{if} represent factor loadings of users and items respectively. For instance, item 1 loads on factor 1 with y_{11}. Note that the original 18-dimensional user-item space is compressed into a two-dimensional representation.

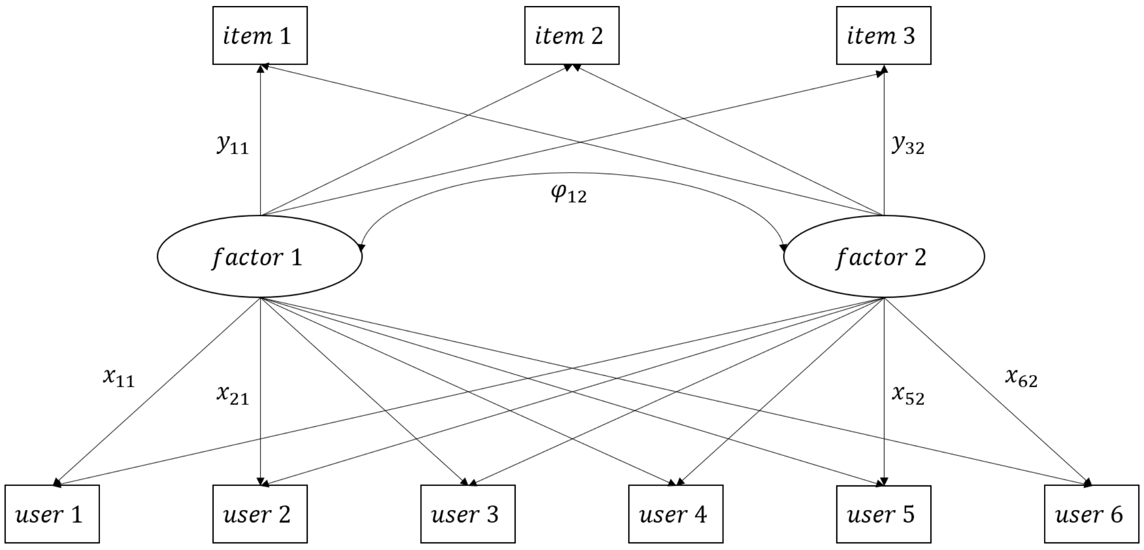


Figure 3: Exemplary path diagram for a collaborative filtering setting with $m = 6$ users and $n = 3$ items which is expressed by $k = 2$ latent factors. Users' factor loadings are denoted by x_{uf}, where u and f represent the number of the user and the latent factor, respectively. Items' factor loadings y_{if} are represented analogously. Factors 1 and 2 may be correlated with correlation coefficient φ_{12} (Source: own illustration after Loehlin and Beaujean (2017, Chapter 5)).

It is important to note that the latent factors remain obscure in a sense that their contentual meaning is never laid bare. Collaborative filtering models merely provide values for loading coefficients but offer no indication on the interpretation of the factors. In some cases operators might get an idea about factors' true nature. In movie recommendation it would be conceivable that factors correspond to movie genres which users have a certain affinity for and items incorporate a certain share of. For meaningful recommendations, however, knowledge about the meaning of factors is irrelevant. Much more important is finding an adequate low-dimensional factor-space representation, which is achieved by matrix factorization (Koren, 2008).

[3]Path diagrams may take on considerably more complex forms than the example displayed here. In particular, inter-item and and inter-user correlations, as well as error terms, have been omitted for the sake of clarity. For detailed explanation of path models see Loehlin and Beaujean (2017).

2.3.2 Matrix Factorization

Motivation

As has been outlined above, latent factor models project user-item interactions into a low-dimensional space by representing both users and items in values of factor loadings. Key to this endeavor is the *factorization* of the ratings matrix R. Matrix factorization is essentially the representation of a matrix by the cross product of two considerably smaller matrices $X^* \in \mathbb{R}^{m \times k}$ and $Y^* \in \mathbb{R}^{n \times k}$ which contain users' and items' factor loadings, respectively. R is thus decomposed such that[4] (Seroussi et al., 2011):

$$R = X^* Y^{*T} \tag{1}$$

Crucially, this decomposition is still viable when R contains missing values, meaning X^* and Y^* can be fully specified from the observed values in R. Consequently, once the factor loadings matrices X^* and Y^* are found, it is possible to rewrite the missing values in R simply as the dot product of vectors of the corresponding column vectors in X^* and Y^*, $x^*_u \in \mathbb{R}^k$ and $y^*_i \in \mathbb{R}^k$ (Yu et al., 2014):

$$r_{ui} = x^{*T}_u y^*_i \tag{2}$$

This way, the exemplary setting from Figure 3 can be expressed as a cross product of factor matrices. Note that users' and items' representation, which in the original space takes n and m values respectively, is compressed into the same $k = 2$ dimensions:

$$R = (\, r_{ui}) = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \\ r_{41} & r_{42} & r_{43} \\ r_{51} & r_{52} & r_{53} \\ r_{61} & r_{62} & r_{63} \end{pmatrix} = X^* Y^{*T} = \begin{pmatrix} x^*_{11} & x^*_{12} \\ x^*_{21} & x^*_{22} \\ x^*_{31} & x^*_{32} \\ x^*_{41} & x^*_{42} \\ x^*_{51} & x^*_{52} \\ x^*_{61} & x^*_{62} \end{pmatrix} \times \begin{pmatrix} y^*_{11} & y^*_{12} & y^*_{13} \\ y^*_{21} & y^*_{22} & y^*_{23} \end{pmatrix}$$

It is easy to see how matrix factorization presents a powerful solution to the recommendation problem. Alas, Equations 1 and 2 are valid only if the true representation (X^*, Y^*) is found. In practice, this is frequently not the case due to the *non-convex* nature of the matrix factorization problem (Jain and Kar, 2017). Rather, solutions X and Y are obtained that possibly do not exactly reflect the true decomposition. Therefore Equations 1 and 2 must in general be rewritten as (see for example Aggarwal, 2016, Chapter 3.6):

$$R \approx XY^T, \qquad r_{ui} \approx x^T_u y_i \tag{3}$$

[4]In many sources, the low-rank matrices are taken to be of dimensions $k \times m$ and $k \times n$ respectively, so $X_{tp} = X^T$ and $Y_{tp} = Y^T$. This leads to the decomposition $R = X^T_{tp} Y_{tp}$. Since this Thesis focuses on the Hu-Koren-Volinsky model, however, it adopts the notation of Hu et al. (2008). Because of $X^T_{tp} Y_{tp} = (\, X^T)\, ^T Y^T = XY^T$ both equations are equivalent.

Optimization Problem

The matrix factorization problem is at heart a *non-convex minimization problem.* The aim is to find $X \in \mathbb{R}^{m\times k}, Y \in \mathbb{R}^{n\times k}$, such that the objective function $J : \mathbb{R}^{m\times n} \to \mathbb{R}$, $(X, Y) \mapsto J((X, Y))$, is minimal[5]:

$$\underset{X,Y}{\arg\min}\, J = \underset{X,Y}{\arg\min} \sum_{u,i\in\Omega} (\, r_{ui} - x_u^T y_i)^2 + \lambda(\, \|X\|_F^2 + \|Y\|_F^2)\,. \tag{4}$$

The first term of J is dedicated to minimizing the squared error of approximating R by XY^T over the set Ω of observed entries of R. The second term is inserted for regularization to prevent overfitting induced by overly large parameters in X and Y. Regularization is carried out by the squared Frobenius norm $\|\cdot\|_F^2$, which is the sum of squared matrix elements. Parameter λ calibrates the severity of punishment (Yu et al., 2014). Regularization is common to modeling but particularly important in the collaborative filtering context where training data, i.e., the observed entries of R, are scarce (Section 2.3.3 will address this problem in more detail). By discouraging large coefficients in X and Y the regularization term inserts a bias towards simpler and thus less data-specific solutions (Aggarwal, 2016, Chapter 3.6).

Minimizing J means finding the function's *global minimum.* In general, functions may possess three types of critical points: local extrema, global extrema, and saddle points. For any function $f : \mathbb{R}^p \to \mathbb{R}^q$, $x \mapsto f(x)$, a local minimum occurs in $x_{min,\,loc}$ if $f(x_{min,\,loc})$ is the minimal functional value in a neighborhood of $x_{min,\,loc}$. The definition of a global minimum $x_{min,\,glob}$ extends this to f taking on the globally minimal value in $x_{min,\,glob}$ (that is, f is minimal in an arbitrarily large neighborhood of $x_{min,\,glob}$). Saddle points x_{sad} mark areas where f merely enters a plateau before decreasing further, not becoming locally minimal. A necessary condition for x_0 being a critical point of any type is that the gradient of f in x_0, $\nabla f(x_0) \in \mathbb{R}^p$, which denotes the vector of f's partial derivatives with respect to $x_0 = (x_{0,1}, x_{0,2}, ..., x_{0,p})$, equals the null vector (Kelley, 1999, Chapters 1.3 and 1.4):

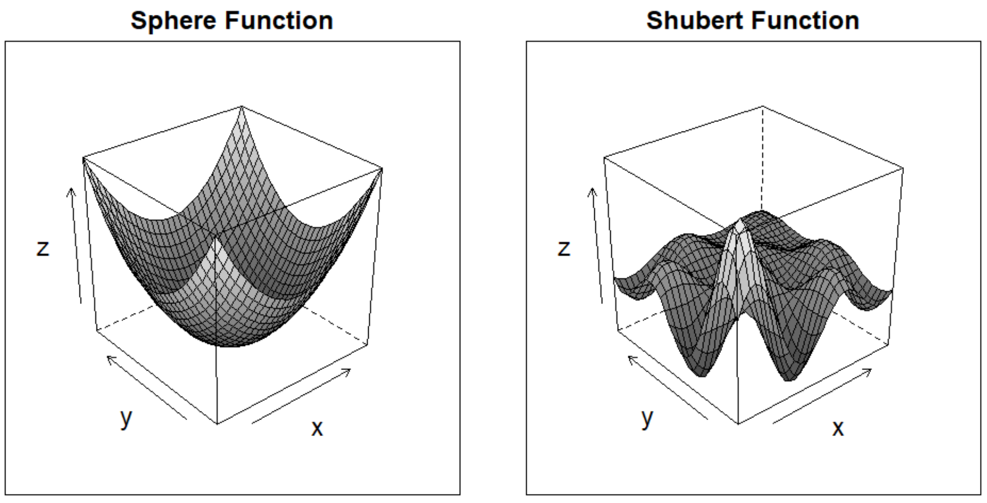
$$\nabla f(x_0) = 0 \tag{5}$$

A gradient of 0 means that, in a critical point, f's slope is neither positive or negative. Whether f decreases before reaching x_0 and then increases again (indicating a minimum), or continues to decrease (indicating a saddle point) can usually be determined by second-order conditions (Kelley, 1999, Chapters 1.3 and 1.4). However, two kinds of ambiguity arise from this situation. First, the distinction between minima and saddle points by applying second-order conditions is not always beyond doubt in optimization settings, since many objective functions have degenerate saddle points that cannot be identified this way (Anandkumar and Ge, 2016). Second, even if a critical point is known to be a minimum, it is not trivial to decide whether

[5]In fact, this minimization expression is somewhat detached from theoretical matrix recovery: since the aim is finding a rank-k representation of R, the corresponding minimum would in theory have to be searched over some $(XY^T)^*$ which is of rank k at most. This poses a constrained optimization problem. However, in practice, the rank constraint is replaced by dimensional limits on X and Y, which leads to the *per se* unconstrained problem of Equation 4. While this deviates somewhat from strict mathematical theory, it works reasonably well for practical applications and allows for substantial computational savings (Bhojanapalli and Neyshabur, 2016).

it is *globally* minimal. This is due to the fact that first- and second-order conditions rely on the differential, which is a local notion and as such confined to local propositions. Finding sufficient conditions for global optimality is not impossible but considerably more complex (Hiriart-Urruty, 1995).

The problem of ambiguity is gravely exacerbated by the fact that the matrix factorization problem stated in Equation 4 is of *non-convex* nature (Jain and Kar, 2017). Unlike convex[6] functions, whose local minima are automatically also globally minimal (Kelley, 1999, Chapter 1.5), non-convex functions may contain numerous local extrema, plus saddle points, where the gradient equals the null vector. Finding even a locally minimal solution to the matrix factorization problem is therefore challenging (Anandkumar and Ge, 2016). Figure 4 gives an idea about the potentially complex topography of a non-convex function (right) as opposed to a convex function (left):



(a) Convex function (b) Non-convex function

Figure 4: Exemplary convex and non-convex bivariate functions. The Sphere function on the left is given by $f : \mathbb{R}^2 \to \mathbb{R}, f(x, y) = x^2 + y^2$. On the right the Shubert function, $f : \mathbb{R}^2 \to \mathbb{R}, f(x, y) = (\sum_{i=1}^{5} cos(\,(\,i + 1)\,ix + i)\,(\sum_{i=1}^{5} cos(\,(\,i + 1)\,iy + i)$, is displayed. Both functions are placed into a cube of side length 2 with $x, y \in [-1, 1]$ (Source: own illustration after Surjanovic and Bingham (2017 (accessed on 2019/06/08)).

Numerical Solutions

Various algorithms can be applied to solve the matrix factorization problem. Aggarwal (2016, Chapter 3.6) provides an extensive overview on the main techniques currently in use for collaborative filtering. They all strive to find a solution for the optimization problem stated in Equation 4 in presence or absence of additional constraints on X and Y. Table 1 lists the main methods, of which unconstrained matrix factorization and singular value decomposition are by far the most popular (see for example Yu et al. (2014) or Koren (2008)):

[6]For a formal definition of convex functions see for example Jain and Kar (2017).

Method	**Constraints**
Unconstrained matrix factorization	None
Singular value decomposition	Orthogonal basis
Non-negative matrix factorization	Non-negativity
Probabilistic matrix factorization	Non-negativity

Table 1: List of main matrix factorization methods and constraints incorporated in their respective optimization problem (Source: own illustration after Aggarwal (2016, Chapter 3.6)).

(a) **Unconstrained matrix factorization** solves the basic optimization problem of Equation 4 without subjecting the optimum to any further constraints. It encompasses two main techniques frequently applied in collaborative filtering settings, namely *stochastic gradient descent* and *alternating least squares* (Yu et al., 2014). However, it must be noted that both methods in their original form fail to account for the unary nature of implicit feedback. Since they have no way of knowing which of the missing values are indeed missing and which must be treated as negative feedback, they inherently take all non-observed entries to be missing or equal zero (Pan et al., 2008). This problem will be addressed in the following section.

(aa) **Stochastic gradient descent (SGD)**. SGD is an iterative algorithm that, from a random starting point, gradually moves along a sequence of ever-smaller values of J. It does so by following the direction of the steepest descent, which is the negative value of the gradient. Thus it steadily approximates the minimum with a step size (or learning rate) of α (Nocedal and Wright, 1999, Chapter 2.2). The k entries in each row x_u of X and y_i of Y are updated using one single observed entry at a time. The observed values are revisited multiple time until the algorithm converges (Aggarwal, 2016, Chapter 3.6). A major problem with SGD is that convergence is greatly impacted by the initial starting point and the learning rate α. Frequently, the algorithm merely converges on local minima or tarries at saddle points for an arbitrarily long time (Anandkumar and Ge, 2016).

(bb) **Alternating least squares (ALS)**. As the name suggests, this algorithm iterates between alternately optimizing X and Y while keeping the respective other matrix fixed. In doing so, ALS reduces the matrix factorization problem to a quadratic one with a closed-form solution in each step. Similar to SGD, no further constraints are imposed on X and Y. Computing one iteration typically takes more time than with SGD, however, in general fewer iterations are needed and row-wise computations can be parallelized (Yu et al., 2014). The Hu-Koren-Volinsky model makes use of ALS optimization, so the algorithm functionality will be presented in detail in Section 3.2.

(b) **Singular value decomposition (SVD)**. SVD performs matrix factorization in presence of constraints: it decomposes R such that the columns of X and Y are mutually orthogonal, i.e. perpendicular to one another (in fact, orthogonality is a strong assumption for real-life settings). With SVD, R is represented by three matrices rather than two. Its rank-k approximation is given by

$$R \approx A\Sigma B^T, \tag{6}$$

where A contains the k largest[7] eigenvectors of RR^T, Σ is a diagonal matrix of the k largest eigenvalues of either RR^T or R^TR, which are identical, and B contains the k largest eigenvectors of R^TR. By convention, X is taken to be $A\Sigma$, which leaves B to equal Y (Ekstrand et al., 2010). In essence, SVD performs optimization of the same objective function as unconstrained matrix factorization but over a smaller set of potential solutions. In collaborative filtering with implicit feedback, where R is not fully specified - SVD too suffers from the problem that missing values are either treated as entirely missing or entirely negative feedback -, this tends to produce higher errors than unconstrained solutions (Aggarwal, 2016, Chapter 3.6).

(c) **Non-negative matrix factorization**. This term refers to a number of matrix factorization algorithms that are constrained by $X, Y \geq 0$. It is obvious how non-negativity applies to collaborative filtering setting with implicit feedback where no negative ratings can be observed. Therefore, while accuracy is not necessarily improved, non-negative matrix factorization provides considerably greater interpretability. However, this advantage is less relevant in settings where the aim is merely finding top-N recommendations without explicit estimation of all unobserved ratings (Aggarwal, 2016, Chapter 3.6).

(d) **Probabilistic matrix factorization**. Lastly, probabilistic techniques have been known to provide adequate solutions to the matrix factorization problem with implicit feedback data. While the aforementioned methods rely on linear algebra, probabilistic matrix factorization incorporates statistical probabilities of users' ratings on items. This close relation to probability theory requires X and Y to be non-negative. Similar to SVD, R is decomposed into three matrices, but with rather different intention. Instead of minimizing error terms the aim is to maximize the predictive power of the model, leading to a decomposition with an intuitive probabilistic interpretation of user behavior (Ekstrand et al., 2010).

2.3.3 Domain-Specific Challenges

Latent factor models face a number of domain-specific challenges that have an impact on recommendation quality. Khusro et al. (2016) have accumulated an extensive list of challenges to recommender systems. In the following, those which are relevant to latent factor models are briefly discussed.

(a) **Sparsity** is arguably the the predominant challenge for latent factor models to tackle, particularly so if feedback data is implicit and negative feedback remains unobserved. Since there are typically large number of users and items,

[7] To be exact, the algorithm would have to be called *truncated* SVD, since only the k largest eigenvectors are used instead of all.

and each user interacts only with very few of these items, the ratings matrix tends to be high-dimensional and extremely sparse at the same time. As has been stated in Equation 4, however, latent factor models rely on minimizing errors with regard to observed entries. Consequently, if those are scarce, the optimization problem is further complicated (Huang et al., 2004). Standard methods such as SGD, ALS and SVD fail to account for the mixed nature of unobserved entries since they are limited to either treating them all as missing values or taking them all to reflect negative values. Obviously, both interpretations are a distortion of reality. Pan et al. (2008) refer to the task of collaborative filtering with unary feedback, where in fact only positive interactions can be known, as *one-class* collaborative filtering. They propose two different strategies to alleviate the one-class problem. The first is to assign different weights to observed and non-observed entries of R, accounting for varying levels of confidence. This strategy is applied by the Hu-Koren-Volinsky model and will be detailed in Section 3.1. Alternatively, sampling methods for randomly classifying some missing entries as negative values are suggested. The experimental studies conducted by Pan and colleagues show that both approaches yield similar results and clearly outperform standard methods which uniformly treat missing values (Pan et al., 2008).

(b) **Scalability** is closely associated with sparsity as both stem from high dimensionality. In most applications, algorithms are required to be easily scalable since they often need to make recommendations in real-time. Besides consuming time, computation comes at a monetary cost since data storage, processing and network resources are often provided by external suppliers (Deelman et al., 2008). Run time until convergence and the portion of computation that can be parallelized or carried out up-front therefore matter greatly. In general, model-based collaborative filtering techniques possess the advantage of scaling only sub-linearly with the number of user-item interactions, as opposed to model-based approaches (Khusro et al., 2016). A detailed discussion of scalability, which must certainly include aspects exceeding statistical analysis, is beyond the scope of this Thesis. For proposals on scalability improvements see for example Takács et al. (2009) or Karydi and Margaritis (2016).

(c) **Cold-start** refers to the challenge of incorporating new users and items with no or very few previous ratings into the system. In such cases, collaborative filtering models tend to produce poor recommendations or cover only a certain amount of user-item interactions (Aggarwal, 2016, Chapter 5.1). As a matter of fact, there is no way of addressing this issue directly for a standard latent factor model. Some models mitigate this shortcoming by incorporating additional information on users and items (Gouvert et al., 2018). Others delegate recommendations for new elements to different types of recommender systems, such as demographics-based or content-based algorithms, effectively converting to a hybrid approach (Khusro et al., 2016).

(d) **Grey sheep** is an issue with similar consequences. The term describes users whose affinities do not match well with those of others in the system. As they rely on item preferences of a community of users when estimating the active user's rating towards different items, collaborative filtering models as a whole tend to provide poor recommendations to users with singular behavior. The grey sheep problem has so far received rather little awareness in research on collaborative filtering, which may be due to the fact that grey sheep users represent only a small number of users overall and are thus not deemed overly important. Recently, some work has been dedicated to this challenge which appears to be a subject of further research (Gras et al., 2017).

(e) **Evaluation** of latent factor models, lastly, poses a permanent problem. The challenge here is two-fold: first, suitable measures for evaluation must be found. The following section will show that classic error metrics, which are frequently used for model evaluation thanks to their simplicity and comparability, are not stand-alone sufficient for assessing the quality of collaborative filtering models. Second, evaluation suffers from a lack of benchmark data against which models can be tested. Such data are mostly kept proprietary because they contain a high amount of sensitive information. Some data sets, though, are openly accessible; these are also often found as test data in academic research (Khusro et al., 2016).

2.3.4 Model Evaluation

As has been stated above, evaluating latent factor models proves challenging. Evaluation is nonetheless crucial to fitting a model that provides satisfactory recommendations. Recommendation quality is not limited to prediction accuracy but a multi-faceted concept that incorporates different quantitative and qualitative aspects, which shall be briefly discussed in the following. Aggarwal (2016, Chapter 7) and Herlocker et al. (2004) suggest the following evaluation aspects:

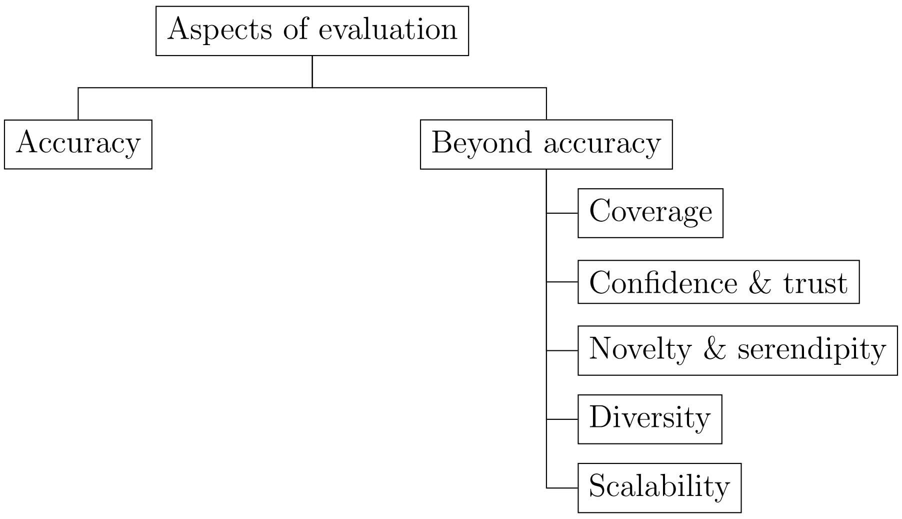


Figure 5: Overview on evaluation aspects for latent factor models (Source: own illustration after Aggarwal (2016, Chapter 7) and Herlocker et al. (2004)).

Accuracy

As latent factor collaborative filtering models essentially perform predictive modeling, a good model is required to achieve high accuracy, that is, produce predictions which reflect the truth as closely as possible. Not least thanks to their simplicity and comparability across different model types, research often relies on error measures to quantify accuracy. In this narrower sense, a model is understood to be fully accurate if all predicted ratings exactly equal the true ones. Accuracy is then evaluated by computing prediction errors such as the *mean absolute error (MAE)* or *root mean square error (RMSE)* (Ekstrand et al., 2010). In collaborative filtering with implicit feedback, however, the aim is to provide top-N recommendations rather than a full estimation of all ratings. Following this concept, accuracy depends on the correct ordering of items with respect to the user's affinity. Accuracy metrics in top-N recommendations thus naturally focus on measures of *rank* (Aggarwal, 2016, Chapter 7.5). Ranking evaluation can be be performed by numerous statistics which can be clustered into correlation, utility and decision-support metrics.

(a) **Correlation-based** rank metrics assess the strength of the relationship between the true and the predicted ranking. Most frequently, ranking accuracy is measured by *Spearman* or *Kendall* rank correlation coefficients. Since both handle weak orderings with many ties poorly, they are not applicable to implicit feedback settings where the unary nature of ratings produces particularly weak orderings (Herlocker et al., 2004).

(b) **Utility-based** metrics seek to quantify the usefulness of recommendations. The underlying assumption is that each recommended item holds a certain utility to the user which is inferred partly by the user's true rating towards it and partly by its position in the recommendation set. Ideally, items with high ground-truth ratings will rank top of the list (Aggarwal, 2016, Chapter 7.5). One way to assess utility is given by the so-called *half-life utility score*. It relies on the assumption that the probability of the user selecting a recommended item from an ordered recommendation list decreases exponentially with the item's position on the list. Let $pos(i)$ denote the position index of item i. The location of the item which has a 50% probability of being examined further is specified by half-life parameter h. Then the half-life utility score R_u for user u is given by[8]

$$R_u = \sum_{i=1}^{n} \frac{r_{ui}}{2^{(pos(i)-1)/(h-1)}} \tag{7}$$

Better recommendation lists will feature higher h-values since utility decreases more slowly. Consequently, the score encompasses more summands with low negative exponents and turns out higher in total. A global utility measure can be computed by averaging the user-specific half-life utility scores over all users (Lü et al., 2012).

[8]The original definition in Lü et al. (2012) has as numerator the maximum out of 0 and the difference between r_{ui} and some default rating. Since implicit feedback is *per se* non-negative and the default rating must certainly be assumed to be 0, Equation 7 has been somewhat simplified.

Alternatively, utility can be measured through the *discounted cumulative gain (DCG)*. The DCG too uses a discount factor relative to the position of items on the recommendation list which is applied to the supposed utility (gain) g_{ui} of user u in consuming item i. Typically, g_{ui} is computed as an exponential function of item i's relevance to user u, rel_{ui}, such that $g_{ui} = 2^{rel_{ui}} - 1$. Relevance is usually approximated by the true rating r_{ui}. With this, the DCG can be expressed as follows[9], where a higher score indicates better recommendations (Aggarwal, 2016, Chapter 7.5):

$$DCG = \frac{1}{m} \sum_{u=1}^{m} \sum_{i=1}^{n} \frac{2^{rel_{ui}} - 1}{\log_2(pos(i) + 1)} \tag{8}$$

There are several other rank-based utility metrics, but since they are mostly derived from the ones explained above, they are not presented in detail. For more information see for example Herlocker et al. (2004).

(c) **Decision-support-based** metrics originate from classification tasks. A model is thought to be accurate if items the user has actually consumed are also deemed worth recommending by the algorithm. For evaluation, a random set of ratings is held out and predicted. In comparing truth and prediction, four distinct cases arise: (1) the algorithm recommends a truly relevant item (true positive), (2) an irrelevant item is recommended (false positive), (3) a relevant item is omitted from recommendation (false negative), or an irrelevant one is not recommended (true negative). Obviously, the second and third constellations represent faulty predictions (Ekstrand et al., 2010). Table 2 illustrates the so-called *confusion matrix* of all cases which contains the respective absolute number of observations for which the stated case applies:

	Relevant	**Irrelevant**
Recommended	# true positive (TP)	# false positive (FP)
Not recommended	# false negative (FN)	# true negative (TN)

Table 2: Confusion matrix for top-N recommendation task. Cells contain the absolute number (denoted by #) of observations for which the respective case applies. Prediction errors occur when irrelevant items are recommended or when relevant ones are omitted from the recommendation list (Source: own illustration after Ekstrand et al. (2010)).

From this confusion matrix several indicators of accuracy can be derived. Let $\hat{R}$ denote a prediction of R. Then the *precision* of the prediction quantifies the share of recommended items that are indeed relevant to the user. With the notation of Table 2 precision is defined as follows:

$$\text{precision}(\hat{R}) = TP/(TP + FP) \tag{9}$$

The *recall* of $\hat{R}$, also referred to as *sensitivity*, measures how many items out of those who are truly relevant to the user have actually been recommended:

[9]In Aggarwal (2016), items are summed only over the set the user has actually rated. With implicit feedback, however, the cases where the user has not provided a rating render a zero summand anyway. For the sake of simplicity this restriction has therefore been omitted.

$$\text{recall}(\hat{R}) = \text{sensitivity}(\hat{R}) = TP/(TP + FN) \tag{10}$$

Lastly, the *false positive rate (FPR)* represents the share of irrelevant items that have been falsely recommended. It is the complement of *specificity*, which measures the share of irrelevant items that have rightly not been recommended:

$$\text{FPR}(\hat{R}) = FP/(FP + TN) \tag{11}$$

The definitions and further information on decision support metrics can be found in Ekstrand et al. (2010). Now an accurate model is expected to have high precision and recall, although those two metrics are somewhat bound by a trade-off. Simultaneously, a low FPR is desirable. Accuracy is often visualized by the so-called *ROC curve*, where ROC stands for *receiver operating characteristic*. It plots sensitivity on the y-axis against FPR on the x-axis and always connects the $(0, 0)$ and $(1, 1)$ coordinates. A model is the more accurate the further its ROC curve deviates from the bisector which would indicate completely random prediction. Therefore, the area under the ROC curve (AUC) is a popular measure for prediction accuracy (Aggarwal, 2016, Chapter 7.5). Figure 6 illustrates an exemplary ROC curve for a rather accurate model:

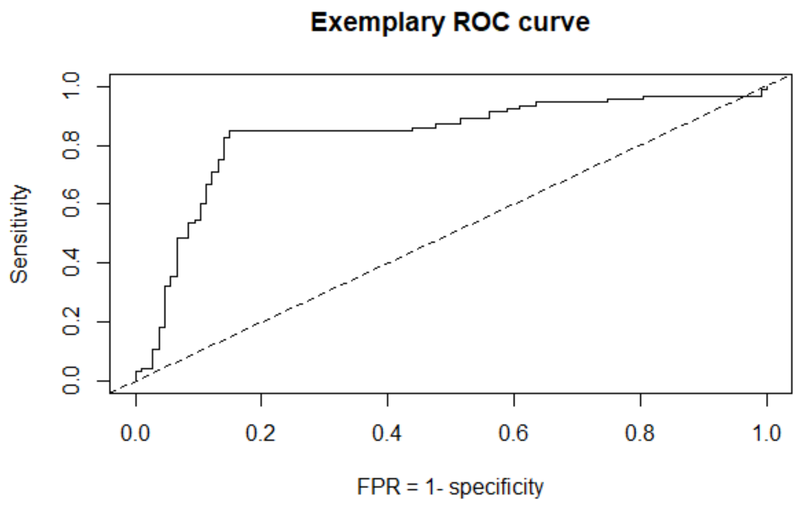


Figure 6: Exemplary ROC curve. The illustrated data are taken from the *ROCR.simple* data set which is implemented in the ROCR package (Sing et al., 2015) and contains mock prediction data. On the horizontal axis, the false positive rate is displayed, while sensitivity is mapped to the vertical axis. The bisector, which represents completely random classification, has been inserted as a dashed line (Source: own illustration).

Beyond Accuracy

Clearly, accuracy is an important quality for latent factor models in collaborative filtering. In revisiting the objectives of recommendations outlined in Chapter 2.1, however, it becomes evident that accurate predictions cannot be the sole criterion. There are several other aspects that should be taken into account but are often hard to measure, which is why accuracy remains the primary field of evaluation.

(a) **Coverage**. The requirement of high coverage is directly linked to the aim of increasing item diversity. Coverage refers to the proportion of user-item interactions the model is capable of predicting and can be divided into two sub-types. *User-space* coverage denotes the share of users for whom at least N recommendations can be derived in a top-N recommendation task, whereas *item-space* coverage measures the fraction of items that can be recommended at all. Both aspects depend heavily on the availability of observed ratings: users for whom little information exists and items consumed rarely pose a problem in being covered (Aggarwal, 2016, Chapter 7.3.2). It is important to note that coverage must always be assessed in relation to accuracy. Boosting coverage by, say, imputing values of average ratings or similar heuristics will impact accuracy negatively (Herlocker et al., 2004).

(b) **Confidence and trust**. These requirements reflect the two perpectives on prediction credibility. *Confidence* measures the degree of certainty the recommender system itself assigns to its predictions, typically expressed as an interval which covers the true value with a fixed probability. Obviously, the system ought to be as confident as possible, in a way that of two equally accurate methods the one with higher confidence is preferable (Shani and Gunawardana, 2011, Chapter 8.3.4). From the complementary perspective, the level of *trust* indicates how much faith users put into the recommendations they receive. Perhaps counter-intuitively, high degrees of accuracy do not necessarily result in users trusting the system. As their preferences towards unknown items often remain latent to themselves, users might not recognize an accurate prediction and be inclined to suspicion. On the other hand, augmenting trust may even be contradictory to some recommendation objectives: for instance, suggesting only already-popular items is likely to preserve trust but will not contribute to item diversity. A possible solution to this dilemma is the display of plausible explanations as to why a specific item is recommended (Aggarwal, 2016, Chapter 7.3.3).

(c) **Novelty and serendipity**. The requirement to suggest *novel* items may also be in conflict with user trust. While novelty is often understood as the proportion of unknown items in the recommended set, Hurley and Zhang (2011) deem this definition too restrictive since it implicitly assumes full information about users' prior knowledge. They propose instead to interpret novelty as items being *unusual* with respect to the user's observed behavior. More precisely, the items that a merely similarity-based algorithm finds hardest to recommend are regarded as most novel (Hurley and Zhang, 2011). Closely linked to novelty is the requirement of *serendipity*, meaning the recommendation should be able to surprise the user in a way that it is unexpected or different from the obvious. It is thus a somewhat stronger concept than novelty (Aggarwal, 2016, Chapter 7.3.5). Suggesting to a user who has been shown to like movies from a certain genre a movie from the same genre she has not yet seen might be novel but unsurprising, whereas recommending a movie from an entirely different genre may be also serendipitous. Kotkov et al. (2016) point out that for a recommender system to achieve serendipity, recommendations must not only be unexpected but also relevant, in a sense that merely provoking surprise is not sufficient when the user does not regard the recommended item as useful.

(d) **Diversity**. Recommender Systems are expected to prove recommendation sets of a certain diversity. Suggesting highly similar items bears the risk of not appealing to the user at all if the prediction turns out to be inaccurate (Aggarwal, 2016, Chapter 7.3.6). Even if the user does approve of the type of items recommended, they might prefer a broader range of options over several variants of one suggestion (Shani and Gunawardana, 2011, Chapter 8.3.8). Furthermore, greater variety is compatible with the requirements of novelty and serendipity.

(e) **Scalability**. The last aspect to be taken into account has been discussed with domain-specific challenges in Chapter 2.3.3. Scalability refers to computational requirements which should obviously be as low as possible since they carry a temporal and, ultimately, monetary cost. As opposed to the aforementioned concepts, scalability can be easily quantified, for instance by calculating up-front training time, prediction time (which is particularly precarious), and memory space (Aggarwal, 2016, Chapter 7.3.8). Some relief to this point may have recently been brought by the advance of cloud computing: instead of hosting their own data infrastructure with development and maintenance costs, suppliers can use cloud services which rent out computing capacity at need (Deelman et al., 2008).

3 The Hu-Koren-Volinsky Latent Factor Model for Implicit Feedback

3.1 Optimization Problem

Now that a basic understanding of collaborative filtering in implicit feedback settings with latent factor models has been established, the Hu-Koren-Volinsky model as one specific proposal to solve the recommendation problem shall be introduced. To this end, the optimization problem and the problem-solving algorithm will be presented in detail. Afterwards, the challenge of hyper-parameter choice will be addressed. All elaborations, unless stated otherwise, are based on the work of Hu et al. (2008).

Original Setting

The Hu-Koren-Volinsky model was initially proposed as a recommender system for television shows. The original model relied on implicit user feedback in the form of anonymous users' watching habits. A rating on show i by user u in this set-up is constituted by the times u has fully watched i. As is characteristic of implicit feedback, this data is asymmetric (no negative feedback can be observed with certainty) and inherently noisy (users might watch a show but dislike it or be absent while it is on).

Incorporation of Confidence

Data asymmetry and noisiness inhibit direct deduction of *preferences* from implicit feedback. However, assumptions may be derived with a certain *confidence*. Not observing a rating at all purports very low confidence; one-time interactions allow for more confidence but might still be distorted due to various reasons; multiple interactions of one user with a specific item corroborate the assumption of preference rather strongly. Hu et al. (2008) incorporate this hypothesis into their model by weighing ratings according to the level of confidence assigned: raw ratings r_{ui} are binarized, yielding assumed preferences p_{ui} which are then paired with a confidence level c_{ui}.

As before, raw ratings r_{ui} are stored in ratings matrix R and can be expressed in the following form:

$$r_{ui} = \begin{cases} j & \text{if } u \text{ has interacted with } i \; j \text{ times,} \qquad j > 0 \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

Ratings may take on floating numbers as they do in the original setting (shows can be watched in part), but in many contexts, such as purchasing or clicking behavior, they are in fact constrained to integers. From these raw numerical values, binary preferences are inferred. Lacking further information, the model declares unobserved interactions as non-preference and observed interactions as preference but later accounts for the respective amount of uncertainty these assumptions carry. Preferences are thus set to:

$$p_{ui} = \begin{cases} 1 & \text{for } r_{ui} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

These binary preferences are precisely what shall be estimated by the model. As before, the aim is to find factor matrices X, Y which constitute a rank-k representation of the original ratings matrix R. Predictions are then computed by multiplying the respective factor loadings vectors of user u and item i:

$$\hat{p_{ui}} = x_u^T y_i \tag{14}$$

Since the direct link of preferences to raw ratings is subject to a starkly varying level of confidence, the degree of uncertainty must be accounted for in making predictions. Confidence scales with the numerical value of raw ratings:

$$c_{ui} = 1 + \alpha r_{ui}, \qquad \alpha > 0 \tag{15}$$

This way, unobserved ratings are assigned a minimum confidence level of 1, while confidence in observed ratings assumes values > 1 that depend on the choice of α. For the latter the authors suggest a parameter value of 40, inserting a substantial discrepancy between respective confidence levels for unobserved and observed ratings. It is easy to see how finding an adequate value for α is crucial to model quality as the incorporation of confidence marks the very core of the Hu-Koren-Volinsky model, which is why Section 4 will address this challenge in more detail.

Objective Function

The assumptions on preference and confidence are inserted into the basic objective function stated in Equation 4 by weighing the inherent prediction error of Equation 2 with the corresponding level of confidence. This marks the first important distinction from the basic function. The second is given by the fact that J_{HKV} is minimized over *all* user-item interactions instead of only the observed ones. While it is acceptable in a general matrix factorization context to confine optimization to observed entries, the fundamental asymmetry of implicit feedback renders this modus operandi inapplicable, since it would equal optimization over positive feedback only. Thus, the optimization problem of the Hu-Koren-Volinsky model is given by

$$\arg\min_{X,Y} J_{HKV} = \arg\min_{X,Y} \sum_{u=1}^{m} \sum_{i=1}^{n} c_{ui} \left(p_{ui} - x_u^T y_i\right)^2 + \lambda \left(\sum_{u=1}^{m} \|x_u\|_F^2 + \sum_{i=1}^{n} \|y_i\|_F^2 \right). \tag{16}$$

In order to keep the following explanations easily traceable, the following notation will be applied henceforth:

$X = (x_{uf})$	$\in \mathbb{R}^{m \times k}$	Matrix of user factors with $u = 1, ..., m,\ f = 1, ..., k$
x_u	$\in \mathbb{R}^{k}$	Vector of factor loadings for user u
$Y = (x_{if})$	$\in \mathbb{R}^{n \times k}$	Matrix of user factors with $i = 1, ..., n,\ f = 1, ..., k$
y_i	$\in \mathbb{R}^{k}$	Vector of factor loadings for item i
$P = (p_{ui})$	$\in \mathbb{R}^{m \times n}$	Matrix of binary preferences with $u = 1, ..., m,\ i = 1, ..., n$
p_u	$\in \mathbb{R}^{n}$	Vector of binary preferences of user u
$C = (c_{ui})$	$\in \mathbb{R}^{m \times n}$	Matrix of confidence levels with $u = 1, ..., m,\ i = 1, ..., n$
c_u	$\in \mathbb{R}^{n}$	Vector of confidence levels for ratings by user u
c_i	$\in \mathbb{R}^{m}$	Vector of confidence levels for ratings on item i

Table 3: Notation for explanations on the Hu-Koren-Volinsky model

3.2 Iterative Solution Algorithm

Minimizing the objective function J_{HKV} over all entries of the highly sparse ratings matrix obviously requires a lot more computation complexity than optimization over only the observed ones would. More precisely, J_{HKV} contains $m \times n$ terms, a number that can easily scale up to billions or trillions of digits. Consequently, the optimization problem reaches such high dimensionality that direct optimization techniques like SGD no longer provide meaningful solutions[10]. Revisiting the exemplary illustration of a non-convex function in Figure 4, it is easily conceivable that finding the global minimum of a function this complex by moving along the gradient is near impossible. Instead of direct optimization the Hu-Koren-Volinsky model therefore relies on an ALS algorithm which, as has been explained in Section 2.3.2, reduces the problem to a quadratic form.

Algorithmic Structure

Broadly spoken, the algorithm takes a random initialization Y_{init} of Y as input in the first step and computes X with fixed Y_{init}, then recomputes Y with fixed X as obtained from the previous step, and iterates over these alternate tasks until convergence is reached. Convergence in this context is thought of as the stabilization in the elements of factor matrices X and Y. Each computation of X and Y is carried out row-wise, that is, by looping over all users or all items, respectively. The algorithm thus roughly assumes the following structure:

[10]This statement is broadly corroborated by related work (see for example He et al. (2019)).

Algorithm 1 Finding X,Y in the Hu-Koren-Volinsky model using ALS

```
initialize $Y$ at random
1: while not(convergence) do
2:   for $u = 1$ to $m$ do
3:     compute $x_u$ using Y
4:   end
5:   yielding $X$
6:   for $i = 1$ to $n$ do
7:     compute $y_i$ using X
8:   end
9:   yielding $Y$
10: end
11: return X,Y
```

First-Order Conditions

Computation of x_u and y_i directly results from the first-order condition of minimization, namely that for $(X, Y)^*$ being at least locally minimal the gradient of J_{HKV} in $(X, Y)^*$ must equal 0. Calculating the partial derivatives of J_{HKV} with respect to x_u and y_i yields the following equations, where C^u and C^i are the diagonal matrices of c_u and c_i, respectively[11]:

$$x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p_u \tag{17}$$

$$y_i = (X^T C^i X + \lambda I)^{-1} X^T C^i p_i \tag{18}$$

Computational Improvements

Applying some transformations to the above conditions allows for saving computational costs. Calculation of $Y^T C^u Y$ and $X^T C^i X$ in each iteration of the user and item loops, respectively, requires a high amount of time. Computation becomes substantially faster when exploiting the fact that

$$Y^T C^u Y = Y^T Y + Y^T (C^u - I) Y \tag{19}$$

and, analogously,

$$X^T C^i X = X^T X + X^T (C^i - I) X. \tag{20}$$

While this may not appear as a simplification at first glance, note that $Y^T Y$ and $X^T X$, which are identical for all users and items, can now be pre-computed before looping over all elements. Moreover, entries of $C^u - I$ and $C^i - I$ are only non-zero if the respective user-item interaction has been observed, i.e the assigned confidence level is > 1, which is true for but a small fraction of entries due to the sparsity of the ratings matrix. The first-order conditions of the optimization problem therefore become:

[11]A more extensive development of these equations from the basic objective function may be found in the Appendix under A.1.

$$x_u = (Y^T Y + Y^T(C^u - I)Y + \lambda I)^{-1} Y^T C^u p_u \tag{21}$$

$$y_i = (X^T X + X^T(C^i - I)X + \lambda I)^{-1} X^T C^i p_i \tag{22}$$

Run times now amount to $O(k^2\nu + k^3 m)$ for computing X and $O(k^2\nu + k^3 n)$ for computing Y (ν denotes the total number of non-zero observations). It becomes immediately clear that computation time scales with the number of users and items, observed entries, and, most notably, the number of factors.

Deriving Recommendations

Eventually, the factor matrices X and Y obtained from factorization of R can be utilized for deriving top-N recommendations. To this end, all preferences are predicted as has been outlined above:

$$\hat{p_{ui}} = x_u^T y_i$$

For each user u, these predicted preferences are then arranged in descending order. The recommendation list is constituted simply by selecting the N items with the highest ranking[12].

Challenge of Hyper-Parameter Choice

As has been mentioned in the context of confidence levels, the degree to which the Hu-Koren-Volinsky model succeeds in factorizing the ratings matrix into a meaningful low-rank representation depends on the adequate choice of certain settings. In general, model parameters that need to be determined by the operator prior to fitting the model, as opposed to those learned by the model during training, are called *hyper-parameters*. Their choice greatly impacts model quality, which is why hyper-parameter optimization remains a field of constant work and discussion, particularly so in research on machine learning (Klatzer and Pock, 2015).

The Hu-Koren-Volinsky model features three hyper-parameters whose values influence recommendation quality. First, the representation of R in a low-rank factor space heavily depends on the number k of factors. Choosing the appropriate number is subject to a trade-off between model fit and simplicity (the latter being also linked to saving computational costs). Second, confidence levels for observations are mainly driven by parameter α. If α is too low, the uncertainty inferred by declaring non-observed interactions as non-preference is not sufficiently accounted for; if α is too high, on the other hand, confidence in observed interactions might be overstated. Third, regularization parameter λ which controls the severity of penalizing complexity must be calibrated.

[12]In practice, the selection of items to be recommended is often subject to further rules and conditions, such as physical availability of the item or profit margin contribution. The incorporation of such additional constraints is also referred to as *context awareness*. For details on this topic see for example Adomavicius and Tuzhilin (2011).

The empirical analysis this Thesis conducts strives to find optimal hyper-parameters for the Hu-Koren-Volinsky model as applied to real data. Results of this endeavour are discussed in Section 5. Before, the following section will lay out the theory of hyper-parameter optimization in more detail, focusing on the two approaches used for the empirical analysis.

4 Hyper-Parameter Tuning in Predictive Modelling

4.1 Principles of Hyper-Parameter Tuning

It has been shown that matrix factorization in latent factor models is in essence an optimization problem. The same is true for the task of hyper-parameter tuning, such that the latter can be interpreted as an outer optimization loop in the entire recommendation problem (Bergstra and Bengio, 2012). It is important to note that hyper-parameters should always be tuned simultaneously as they are frequently co-dependent. In the following, the dominant approach to hyper-parameter optimization is laid out as found in Bergstra and Bengio (2012).

Optimization Problem

Typically, a learning algorithm, such as the one enshrined in the Hu-Koren-Volinsky model, strives to find a function f that minimizes some loss function $\mathcal{L}(x; f)$, where x is assumed to follow ground-truth distribution $\mathcal{G}_x$. The learning algorithm is itself dependent on the choice of hyper-parameter set θ and can thus be expressed as $\mathcal{A}_\theta$. The function minimizing the expected loss over training data $\mathcal{X}_{\text{train}}$ is then given by $f = \mathcal{A}_\theta(\mathcal{X}_{\text{train}})$. Consequently, the set θ^* of optimal hyper-parameters is obtained by minimizing the expected loss over the set Θ of all possible combinations (Bergstra and Bengio, 2012):

$$\theta^* = \underset{\theta \in \Theta}{\arg\min}\, \mathbb{E}_{\mathcal{G}_x} \left(\mathcal{L} \left(x; \mathcal{A}_\theta(\mathcal{X}_{\text{train}}) \right) \right) \tag{23}$$

The exact form of loss function f thereby depends on which model selection criterion is applied to distinguish good models from poor ones. This decision is directly linked to model evaluation as discussed in Section 2.3.4, so, for instance, the ideal set of hyper-parameters could be established through optimization of half-life utility or AUC. Due to the rather elusive nature of beyond-accuracy measures, hyper-parameter optimization usually relies on easily quantifiable accuracy metrics (Luo, 2016).

Approximate Solution via Cross Validation

Finding θ^* as stated in Equation 23 faces several obstacles to a direct solution. The first is that ground-truth distribution $\mathcal{G}_x$ is unknown, which renders the exact computation of expected value $\mathbb{E}_{\mathcal{G}_x}$ impossible. Therefore, most applications use the technique of *cross-validation* for approximating the expected value. In cross-validation, validation sets $\mathcal{X}_{\text{valid}}$ are sampled randomly and then used for computing the mean loss function. If validation sets are indeed drawn independently, no bias is induced. The optimization problem with cross-validation can be written as follows, simplifying the expression by using Ψ for denoting the hyper-parameter response function (Bergstra and Bengio, 2012):

$$\theta^* \approx \underset{\theta \in \Theta}{\arg\min} \underset{x \in \mathcal{X}_{\text{valid}}}{\text{mean}} \mathcal{L}\left(x; \mathcal{A}_\theta(\mathcal{X}_{\text{train}})\right) = \underset{\theta \in \Theta}{\arg\min} \Psi(\theta) \tag{24}$$

Confinement of the Search Space

While ignorance of the ground-truth distribution can be mitigated by cross-validating the estimations of θ^*, the critical point in hyper-parameter optimization is the usually very limited knowledge about the *response surface* $\Psi(\theta)$ and the *search space* Θ. Therefore, it has become the dominant strategy to confine Θ to a small, finite subset Θ_{trial} out of which all distinct θ are tested. The final optimization problem as encountered in practical applications thus takes the form:

$$\theta^* \approx \underset{\theta \in \Theta_{\text{trial}}}{\arg\min} \Psi(\theta) = \hat{\theta} \tag{25}$$

Obviously, finding Θ_{trial} such that it contains only few promising values for θ is a non-trivial task which becomes all the more complicated with higher numbers of hyper-parameters (Bergstra and Bengio, 2012). The following section will contrast the two most widely used approaches for confining the search space that are applied for tuning the hyper-parameters of the Hu-Koren-Volinsky model in the empirical analysis.

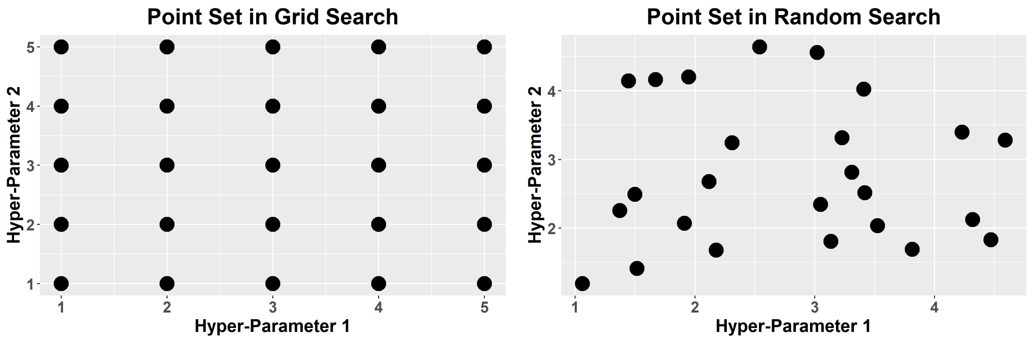
4.2 Two Selected Approaches

4.2.1 Grid Search

Grid search uses a rather simple method to construct a search space for the optimization algorithm. Essentially, this approach encompasses the definition of a configuration space for each hyper-parameter and the consequent evaluation of all possible combinations, the globally optimal of which is then used to fit the model. Parameter combinations are spread out on a discrete grid that takes the form of a hyper-rectangle (Fu et al., 2016). While implementation and parallelization are easily accomplished, grid search suffers from the *curse of dimensionality.* Let Θ_s denote the configuration space of hyper-parameter s, $s = 1, ..., S$. Then the number of trials to be performed scales to $|\Theta_{\text{trial}}| = \prod_{s=1}^{S} |\Theta_s|$, growing exponentially with rising number S. It is easy to see how this relation limits grid size in practice, which in turn weighs on the probability of the grid containing sensible hyper-parameter values (Bergstra and Bengio, 2012). However, grid search tends to perform reasonably well in settings with few hyper-parameters where the grid remains relatively small (Fu et al., 2016). Since the Hu-Koren-Volinsky model contains but three hyper-parameters, the approach is considered to be worth testing.

4.2.2 Random Search

Random search differs from grid search in proposing an alternative strategy for obtaining Θ_{trial}. Rather than incorporating all possible combinations arising from the configuration space given by the Cartesian product of sub-configuration spaces Θ_s, it draws from a uniform density on this same configuration space. The underlying idea is that of *low effective dimensionality* of Ψ, meaning Ψ is in fact dominated by a small number of important dimensions. Grid search covers the original S-dimensional space evenly but provides inefficient coverage of sub-spaces, because points along one dimension of a grid share the same corresponding coordinate. By contrast, randomly distributed points assume more distinct values in each sub-space. Consequently, for the same number of computationally costly trials, random search manages to test more distinct parameter values (Bergstra and Bengio, 2012). Figure 7 illustrates this situation:



(a) Grid Search (b) Random Search

Figure 7: Exemplary illustration of point sets in grid search and random search. Optimization takes place over two hyper-parameters. In grid search, configuration spaces for both hyper-parameters are given by $\Theta_s = \{1, ..., 5\}$ with $s \in \{1, 2\}$. For random search 25 points are randomly sampled from a uniform distribution with limits $[1, 5]$. It becomes visible that with grid search, 25 trials test only five distinct values of each parameter. By contrast, random search allows for up to 25 distinct values (Source: own illustration after Bergstra and Bengio (2012)).

So, without forsaking the advantages of easy implementation and parallelization, random search covers the search space much more effectively. In fact, discussions suggest that quite a small amount of observations might be enough to find a solution close to the optimum with high probability. For instance, randomly drawn observations have an individual chance of 5% of belonging to the 5%-interval around the true optimum. Drawing n points independently leads to a probability of all points missing this interval of $(1 - 5\%)^n$. Consequently, the chance of at least one of them belonging to the close-to-optimality interval is $1 - (1 - 5\%)^n$. Demanding a probability of at least 95% for this to happen equals n being ≥ 60. So, if at least 5% of the points on the grid represent a solution deemed sufficiently close to the optimum, then random search with 60 trials will identify one point out of that region with a 95% probability. Obviously, higher numbers of trials allow for more certainty (Zheng, 2015 (accessed on 2019/27/08).

However, just as grid search, random search is a *non-adaptive* approach: there is no utilization of already available results for a sensible confinement of the search space. This shortcoming can be mitigated by performing upstream manual search, that is, identifying promising regions of Θ with the help of human intuition or experience (Young et al., 2015).

The following section will now discuss concepts and results of the empirical analysis. As has been mentioned before, the analysis essentially encompasses the implementation of a Hu-Koren-Volinsky model for real-world data with special emphasis on hyper-parameter tuning. First, the concept of analysis is introduced in Section 5.1, including information on the data set used, a brief overview on methodology, and an explanation of the applied evaluation measures. Results are presented in a comparative analysis of the selected approaches towards hyper-parameter optimization. Eventually, these results are discussed with regard to evaluation of both methods and possible enhancements to the fitted model which are beyond the scope of this Thesis but which might help to improve recommendation quality.

5 Fitting the Model With Optimized Hyper-Parameters

5.1 Concept of Analysis

5.1.1 Data Used

In predictive modelling with hyper-parameter optimization, models are usually fitted on training data and evaluated on test data (see Section 4.1). Typically, folds to serve as test data are drawn randomly from the entire data set (Kohavi, 1995). However, this approach is not applicable to implicit feedback as the predicted variable is unknown for all but a few observations and exploiting correlation structures across all elements is crucial (Hu et al., 2008). Therefore, rather than splitting the same data set into different folds multiple times, the data is divided once into two subsequent time periods, such that the quality of recommendations based on the first period can be measured against actual transactions from the second period. Partitioning the data this way is conceivable in any proportion but carries an inherent trade-off. While an overly large test data set risks sub-optimal model fit due to lack of learning data, an over-emphasis on training data might cause overfitting and leave evaluation unreliable. In practice, reserving around 30% of data for testing has often proven effective (Liu and Cocea, 2017).

In this analysis, data is collected from real-world transactions of a Germany-based retail company with international operations[13]. The training set contains purchasing data from the nine-month time period between September 1st, 2018, and May 31st, 2019, whereas transactions from the subsequent three-month period between June 1st, 2019, and August 31st, 2019, are used as test data. Both parts encompass the same sets of users and items, respectively. In effect the train-test split thus takes on a 75%-25% ratio.

Users and Items Included

Since the company features great numbers of both users and items, processing the entire set of available transactions is not feasible without considerable computation power which, to this scale, is not accessible here. Therefore, for this Thesis, an extract of purchasing data is usedö. This extract is selected according to the following criteria:

[13]In order to uphold the ability to publish this Thesis for any audience, no information which might reveal the company's identity is disclosed.

- **Comparability**. The company has evidence suggesting that user clienteles vary across the countries the company operates business in, due to demographic and regional idiosyncrasies. Likewise, item assortments are country-specific. Since latent factor models seek to exploit correlations across users and items, though, it appears sensible to restrict recommendations to user and item groups deemed sufficiently similar for meaningful correlation-based recommendations. Therefore, only purchases of items offered in Germany by users registered in Germany are taken into account.

- **Activity**. Users and items included are selected based on transaction activity, that is, only those users and items with a certain amount of purchases in the considered time period are covered. This approach reflects reality since, as has been argued above, recommendation quality tends to deteriorate with mounting data sparsity. For users and items with very few transactions (or even none at all), other methods than latent factor models are often more advisable.

- **Applicability**. Some groups of users and items need to be excluded as they are not fit to produce sensible recommendations. Concerning the former, these are users with an implausibly high number of transactions (due to fraudulent use of user accounts, as the company's information suggests), so only users with a maximum of 50 purchases in the regarded time period are selected. This corresponds roughly with at most weekly purchases, which is plausible in the company's business environment. Also items from certain item categories that are not applicable for recommendations are omitted.

- **Processability**. The total number of users and items is substantially restricted by the available computation capacity. For this analysis, dimensions of just under 30 million elements in the ratings matrix have been found to work sufficiently well (larger dimensions cause problems with memory space; details on processing infrastructure and methods may be found in the following section).

- **Proportionality**. The exact numbers of users and items are determined according to the company's actual ratio of registered users to items on offer, which is roughly 3.5, so, in total, the 10,000 most active users and 3,000 most frequently purchased items are selected[14].

Due to the absence of some interactions out of the Cartesian product of these sets in the regarded time period, the resulting ratings matrix is slightly smaller than the maximum. With a total amount of 9,766 users and 2,993 items taken into account, it contains 29,229,638 elements (see Table 4). Note how the curse of dimensionality comes into effect here: the already considerable size of the input matrix arises from selecting a few thousand users and items, numbers that are dwarfed by the amounts of data real-world applications - relying on vast amounts of storage and computation capacity - need to process.

[14]In order to ensure model applicability throughout the time period regarded, only users that had been enrolled during the entire time are eligible. Similarly, only products that had consistently been part of the assortment are covered.

Effective number of users	9,766
Effective number of items	2,993
Total effective number of matrix elements	29,229,638
Time period for training	Sep 1st, 2018 to May 31st, 2019
Time period for testing	Jun 1st, 2019 to Aug 31st, 2019

Table 4: Summary of information on the input data sets

Form of Data Input

The raw training and test data sets are extracted from the company's data warehouse (the corresponding query may be found in the Appendix under Section A.2). Both are of identical shape and contain one row per transaction, that is, per purchase of item i by user u. Users and items are represented by their respective IDs, such that each transaction consists of a tuple $(\mathrm{UID}_u, \mathrm{IID}_i)_s$. Note that a tuple may appear repeatedly if u purchased i multiple times within the period considered. Table 5 shows the first five rows of the training data set as an example:

UID	**IID**
lSV+IWuYeCd6ZXSdWjV7UOopYUqPPyRAJiMmkCdFKZlvP2ZV	491549592
lSR3JmYjdSR6aHYtUzV7ncAqYUpAPitAkZIrOSJDkZ9jOVNl	492096516
JCFDIlcmQpd6b3EuZjV7S7kiYXSKj5hAJZUoiyZykZtoQFVm	491293926
lyN0LmiWcyF6Y0icYzV7Tux3YUo6O5ZAIpUlOZh3nithimRn	491794006
LSN0KGWUd5V6ZXQrZTV7oPAkYXM5MyFAmCSYPJZznZtVj1Jk	492086821

Table 5: Extract of the training data set

Descriptive Attributes of Data Input

In order to obtain a more in-depth picture of the data structure and also to identify potential outliers or implausible observations, both data sets shall be examined more closely with respect to their major statistical characteristics. As the Hu-Koren-Volinsky model takes the matrix of raw ratings as input, tuples of transaction data are processed in a way that the absolute frequency of each transaction $(\mathrm{UID}_u, \mathrm{IID}_i)$ is represented by the corresponding cell $r_{s,ui}$ in R_s ($s \in \{\text{training}, \text{test}\}$). Table 6 lists the main attributes of training and test ratings.

Data	**Min**	**Mean**	**Median**	**Max**	**STD**	**Sparsity**
$R_{training}$	0	0.00336	0	50	0.09552	99.668%
R_{test}	0	0.00265	0	1	0.05149	99.738%

Table 6: Major statistical attributes of training ($R_{training}$) and test (R_{test}) data . *Min* is short for minimum, *Max* for maximum, and standard deviation is abbreviated by *STD.*

As expected, the minimum rating in both data sets is 0, equalling a non-observed purchase by user u of item i. The maximum rating of 50 in the training set appears surprising at first but is not implausible since many items included have a higher purchasing frequency. While such high values should certainly be regarded as outliers - the median of 0 suggests their exceptionality - no need for deletion or manipulation can be identified. The maximum rating of 1 in the test data, on the other hand, remains fully within the expected spectrum. Mean and standard deviation reflect the difference in value ranges of training and test data. Lastly, it becomes clear how very sparse the data sets are: non-zero elements account for less than 0.5% in both cases. The fact that the test matrix is even sparser than the training matrix can conceivably be attributed to the shorter time period the former stems from (i.e., less time for transactions to occur).

Drilling down on the two dimensions of the data reveals the distribution of purchases across users and items. In Figures 8 to 13 below the different distributions are displayed. Red and taupe colors represent training and test data sets, respectively. Distributions across users are shaded darker, whereas light colors mark the items dimension. Since single items have larger maximum purchasing frequencies than single users and their distribution is more strongly skewed, lower numbers being more probable, two types of histogram are used to depict the items dimension. The first one (Figures 10 and 11) shows the purchase distribution across the whole range of values and the second one (Figures 12 and 13) zooms into the denser region of zero to 100 purchases.

The number of purchases per user takes on a similar shape in training and test data (Figures 8 and 9). Both distributions are skewed towards lower values and concentrated around a mode of six purchases, with a tail reaching to 50 purchases in the training and 28 in the test set. As the test data result from a shorter time period, their somewhat higher density in lower values is plausible.

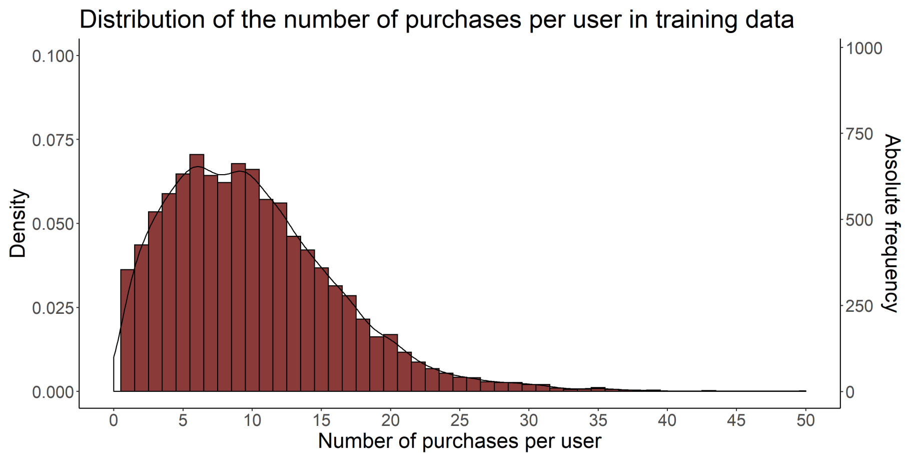


Figure 8: Distribution of the number of purchases per user in training data. The horizontal axis represents the number of purchases. Bars display the absolute frequency of users with the corresponding number of purchases (right vertical axis), the line depicts density (left vertical axis).

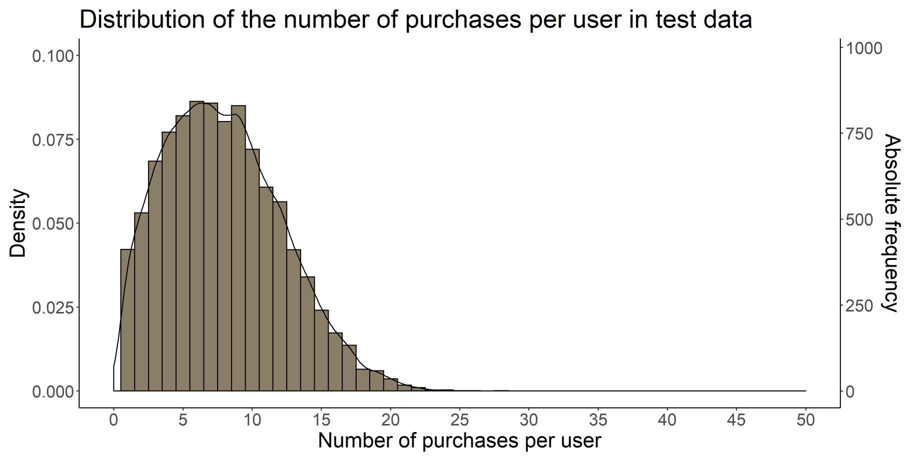


Figure 9: Distribution of the number of purchases per user in test data. The horizontal axis represents the number of purchases. Bars display the absolute frequency of users with the corresponding number of purchases (right vertical axis), the line depicts density (left vertical axis).

Regarding the items dimension, a much greater variance in the number of purchases becomes visible. Again, distributions for both training and test data are strongly skewed to the left with some outliers far in the area of larger values. Considering the variety of items included, some being high-frequency products and others having a long product life cycle, this remains fully within the expected range.

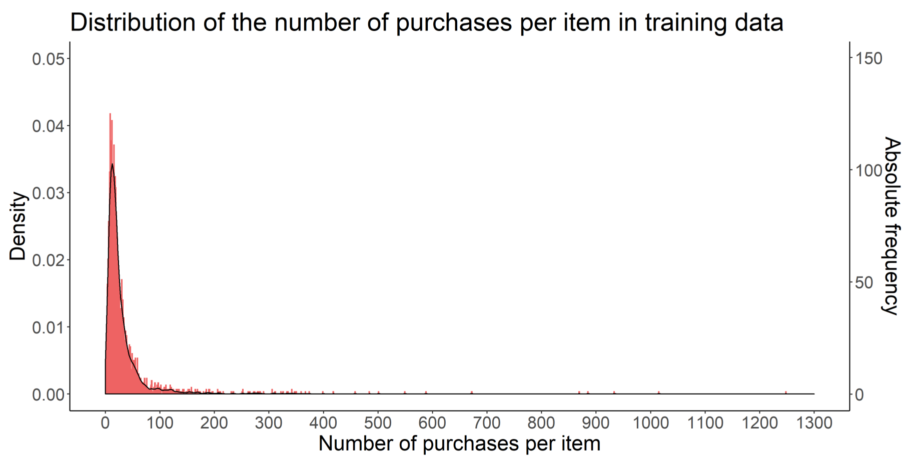


Figure 10: Distribution of the number of purchases per item in training data. The horizontal axis represents the number of purchases. Bars display the absolute frequency of items with the corresponding number of purchases (right vertical axis), the line depicts density (left vertical axis).

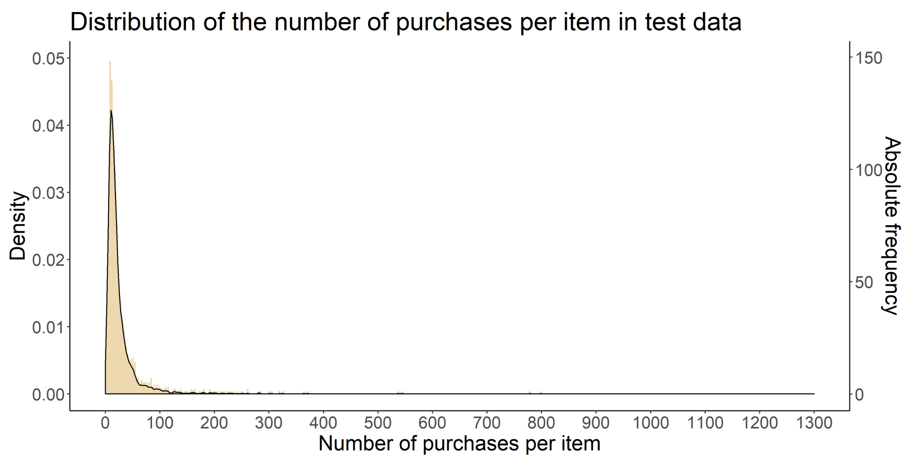


Figure 11: Distribution of the number of purchases per item in test data. The horizontal axis represents the number of purchases. Bars display the absolute frequency of items with the corresponding number of purchases (right vertical axis), the line depicts density (left vertical axis).

In order to get a more detailed picture about the lion's share of density, the two figures below confine the range of purchases to a maximum of 100. Training and test data show similar characteristics, with purchase numbers of around ten being most frequent. Due to the shorter time scope, here too the test set's distribution is even more heavily skewed leftwards.

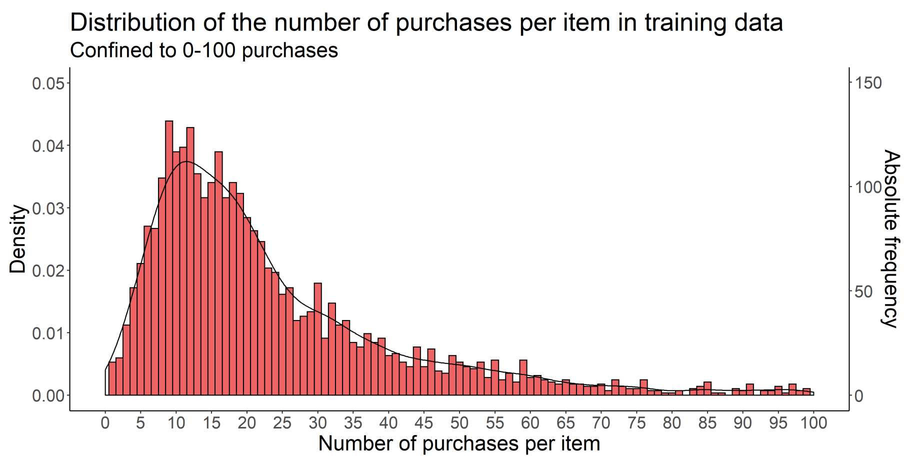


Figure 12: Distribution of the number of purchases per item in training data, confined to 100. The horizontal axis represents the number of purchases. Bars display the absolute frequency of items with the corresponding number of purchases (right vertical axis), the line depicts density (left vertical axis).

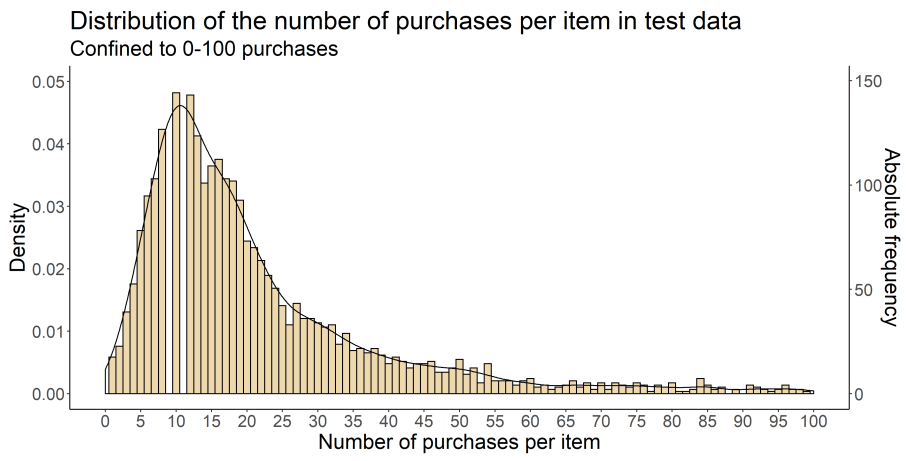


Figure 13: Distribution of the number of purchases per item in test data, confined to 100. The horizontal axis represents the number of purchases. Bars display the absolute frequency of items with the corresponding number of purchases (right vertical axis), the line depicts density (left vertical axis).

All in all, the number of purchases across users and items is distributed in a plausible form and does not point to any irregularities or faultiness of data. As expected, single users and items predominantly feature only relatively small amounts of purchases, which might prove challenging to recommendations. Section 5.3 will discuss how well the model copes with this data sparsity. In the following, the methodological process for fitting and evaluating the model is laid out in detail.

5.1.2 Methodology

The analytical process encompasses several steps that are classic to predictive modeling (see for example Aggarwal (2016, Chapter 2.2)), listed below. All statistical computations are carried out in the software R (R Core Team, 2019). Besides standard R language, the following programming packages are used: *tidyr* for processing data in a tidy manner (Wickham and Henry, 2019), *dplyr* for facilitating data manipulation (Wickham, François, Henry and Müller, 2019), *Matrix* for storing large, sparse matrices efficiently (Bates et al., 2019), *Rfast* for performing fast calculations (Papadakis et al., 2019), *foreach* for parallelizing computations (Wickham, Chang, Henry, Pedersen, Takahashi, Wilke, Woo and Yutani, 2019), *doParallel* as backend for *foreach* (Ooi et al., 2019), *ggplot2* for flexible graphics (Ooi and Weston, 2019), *plotly* for visualizing three-dimensional data (Sievert et al., 2019), and, lastly, *ROCR* for visualizing ROC curves (Sing et al., 2015). All major procedures are self-implemented so as to ensure sufficient flexibility for customization. To enhance feasibility despite the resulting larger computational requirements, procedures are parallelized as far as possible. For the same purpose computations are carried out on the Linuxcluster of the *Leibniz Supercomputing Center of the Bavarian Academy of Sciences and Humanities (LRZ)*, employing eight central processing units. In the following, the analytical scheme is laid out step by step.

[1] **Pre-process data**. Pre-processing in this case is not cumbersome and merely consists of transforming the input data sets into ratings matrices $R_{training}$ and R_{test}. Since both are very sparse, as shown above, they can be stored in a memory-sparing way logging only the non-zero matrix elements.

[2] **Split data in to training and test sets**. As has been explained above, the nature of implicit feedback requires partitioning the data *ex ante*.

[3] **Fit model with optimized hyper-parameters**

[3.1] **Determine trial points for hyper-parameters.** As discussed in Section 4.2, grid and random search serve to confine the search space to a finite number of trial points Θ_{trial} which can be tested for hyper-parameter optimization. However, both require the initial specification of ranges for each hyper-parameter over which to form the search grid or from which to draw random trial points. This endeavor is largely driven by trial-and-error. In the original setting, the authors hint at which values for the number of factors, α, and λ, respectively, work best for their application. These values, though strongly dependent on the concrete data situation, are taken as naive starting points:

- **Number of factors**. This hyper-parameter greatly impacts computational requirements, which is why the number of factors is capped by computational capacity. In Hu et al. (2008), the authors test values ranging between ten and 200, albeit for a substantially larger data set with millions of non-zero entries alone. They find larger numbers of factors to perform better (Hu et al., 2008). Since the data examined here is of much lower dimension and computation power is presumably restricted even more, values from ten to 20 factors are tested.
- **Confidence parameter** α. Here the authors merely mention that $\alpha = 40$ yields satisfying results (Hu et al., 2008). In the attempt to cover a large area around this point, the test range for α is set to [10, 100].
- **Regularization parameter** λ. The only hint about a sensible value of λ refers to a model without confidence adjustment that the authors use as baseline for their proposal (Hu et al., 2008). Lacking other

plausible suggestions, λ is also tested in values ranging from ten to 100. Table 7 summarizes the respective lower and upper bounds for all three hyper-parameters.

Hyper-parameter	**Lower bound**	**Upper bound**
Number of factors	10	20
α	10	100
λ	10	100

Table 7: Search space for hyper-parameter tuning. The three hyper-parameters over which the model is optimized - the number of factors, confidence parameter α and regularization parameter λ are displayed along with the respective lower and upper bounds of the range from which their values have been tried.

The shape of the resulting search grid in grid search is strongly determined by the number of factors which must obviously take on an integer value. Therefore, eleven distinct values (length of the range from ten to 20) for each hyper-parameter are tested, such that, in total, grid search is performed over $11^3 = 1,331$ combinations. Since random search is expected to cover the search space more efficiently, a smaller number of trials should suffice here. The amount of trials is set to 266, which corresponds with 20% of trials in grid search.

[3.2] **Fit model on training data for each trial combination of hyper-parameters and evaluate model performance.** In this step, the Hu-Koren-Volinsky model (without alterations to the form originally proposed in Hu et al. (2008)) is fitted for each hyper-parameter combination in $\Theta_{trial,GS}$ from grid search and $\Theta_{trial,RS}$ from random search, respectively. Since the inherent ALS algorithm can *per se* be executed indefinitely, stopping criteria which cause the factorization to terminate are applied. For the execution to abort, either a critical value of the model evaluation measure (30%) must be reached or the a maximum number of five iterations must be completed. Model evaluation adopts the originally proposed evaluation measure by Hu et al. (2008). Neither the original setting nor the one examined here offers the opportunity to assess users' reaction to recommendations (with the exception of a purchase, whose absence, as has been argued, can stem from a variety of reasons), and, consequently, quantify prediction error. Error-based metrics thus do not serve the evaluation purpose well. Instead, a form of decision-support-based measure is applied. The *average rank* ($\overline{rank}$) is akin to the concept of recall and assesses how well the items actually consumed by a user are represented in the recommendation list. For each user-item interaction, $rank_{ui}$ denotes the percentile-rank of item i in user u's recommendation list. Values of $rank_{ui}$ range between 0% for the topmost and 100% for the least recommended item, where random predictions have an expected $rank_{ui}$ of 50%. Percentile-ranks are then weighted by users' real ratings r_{ui} and aggregated over all users and items, such that only ranks on actually consumed items (where $r_{ui} > 0$) are included and lower values of $\overline{rank}$ are desirable as these indicate that actually consumed items are located further up the recommendation list (Hu et al., 2008):

$$\overline{rank} = \frac{\sum_{u=1}^{m} \sum_{i=1}^{n} r_{ui} rank_{ui}}{\sum_{u=1}^{m} \sum_{i=1}^{n} r_{ui}} \tag{26}$$

The denominator places $\overline{rank}$ into a $[0, 1]$ interval where low values are preferable and 50%, corresponding to random prediction, acts as benchmark for model quality. Figure 14 illustrates the functionality of $\overline{rank}$ for two different models A and B, both of which provide recommendations on a total of nine items for a single user u. True ratings r_{ui} are sorted in descending order and models' rankings $rank_{ui,A}$, $rank_{ui,B}$ are displayed alongside. It is clearly visible that model A shows a better recall - it only confuses items 2 and 3 while model B mis-ranks several items. Consequently, $\overline{rank_{u,A}}$ equals 17.5% but model B scores rather poorly with $\overline{rank_{u,B}} = 35\%$:

Item i	r_{ui}	$rank_{ui,A}$	$rank_{ui,B}$	$r_{ui} \cdot rank_{ui,A}$	$r_{ui} \cdot rank_{ui,B}$
Item 1	4	0.000	0.500	0.000	4.000
Item 2	3	0.250	0.125	1.500	1.500
Item 3	1	0.125	0.000	0.125	0.000
Item 4	1	0.375	0.625	0.500	0.500
Item 5	1	0.500	0.500	1.000	1.000
Item 6	0	0.625	0.250	0.000	0.000
Item 7	0	0.750	0.750	0.000	0.000
Item 8	0	0.875	0.875	0.000	0.000
Item 9	0	1.000	1.000	0.000	0.000
SUM	10			1.750	3.500

$$\overline{rank_{u,A}} = \frac{\sum_{i=1}^{9} r_{ui} rank_{ui,A}}{\sum_{i=1}^{9} r_{ui}} = \frac{1.75}{10} = 0.175$$

$$\overline{rank_{u,B}} = \frac{\sum_{i=1}^{9} r_{ui} rank_{ui,B}}{\sum_{i=1}^{9} r_{ui}} = \frac{3.5}{10} = 0.35$$

Figure 14: Exemplary calculation of average rank measure. Column r_{ui} lists true user ratings on items 1-9. The two subsequent columns contain models' respective rankings on these items. The product of user ratings and model rankings is calculated in the last two columns, the respective sum of which marks the counter in the $\overline{rank_{u,X}}$ ($X \in \{A, B\}$) measure. Dividing by the sum over r_{ui} yields the respective $\overline{rank_{u,X}}$ values (Source: own illustration).

[3.3] **Select best models.** After fitting all possible models with trial points from grid and random search, the respective best model from both approaches is determined according to the lowest achieved average rank.

[4] **Examine best models.** Eventually, the models selected in Step 3.3 are examined more closely in order to assess their performance, providing a foundation for the discussion in Section 5.3.

5.2 Results

5.2.1 Results of Grid Search

After trying 1,331 different combinations of the three hyper-parameters to be specified (making up $\Theta_{trial,GS}$), grid search returns the locally optimal Hu-Koren-Volinsky model with

- 20 factors,
- confidence parameter $\alpha = 100$, and
- regularization parameter $\lambda = 10$.

Figure 15 displays all hyper-parameter combinations constituting $\Theta_{trial,GS}$, each represented as a point in the three-dimensional sub-space of Θ yielded by confining the search space to the examined ranges. The grid-like structure of the search space is easily visible. Points are colored according to the average rank measure of the specified model, where darker colors indicate lower average rank values and thus better models. It becomes clear that well-performing models are concentrated in the area of high factor numbers, high α and low λ. The point representing the locally optimal model is located in the foremost corner on the plane marking $\lambda = 10$.

The model achieves satisfying results with an average rank of 6.83%, remaining well below the benchmark of 50% expected from random recommendations. This means that items actually purchased in the test phase are on average located within the top 6.83% of the recommendation list, indicating that the model is capable of grasping users' affinities quite well.

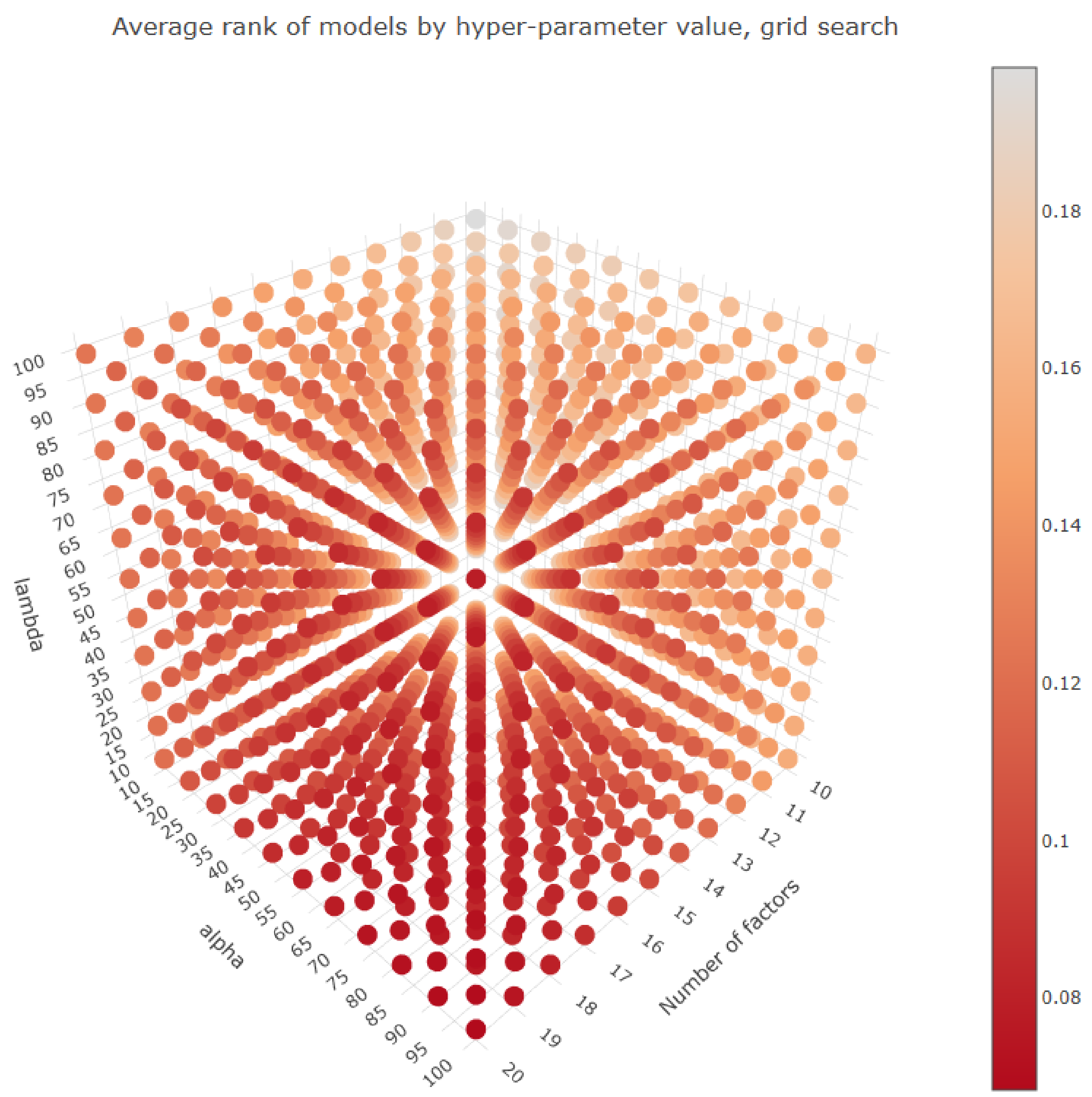


Figure 15: Average rank of models by hyper-parameter value as obtained by grid search. The three hyper-parameters factor amount, α and λ are mapped to the x-axis, y-axis and z-axis, respectively. Each point in the thus constituted space represents a different model with corresponding hyper-parameter values. Darker colors indicate lower average rank values and therefore better models.

Addressing accuracy from another angle, the ROC curve of the locally optimal model, as obtained by grid search, is displayed in Figure 16. As has been argued before, collaborative filtering with implicit feedback is essentially a classification problem. Interpreting the binarized preferences indicated by the presence or absence of a purchase in the test phase as classes *relevant* and *irrelevant*, and ranks assigned by the model as an estimation for class membership, allows for construction of the ROC curve and the corresponding AUC measure. The visualization suggests that the model achieves near perfect classification, which is corroborated by the high AUC value of 93.35%.

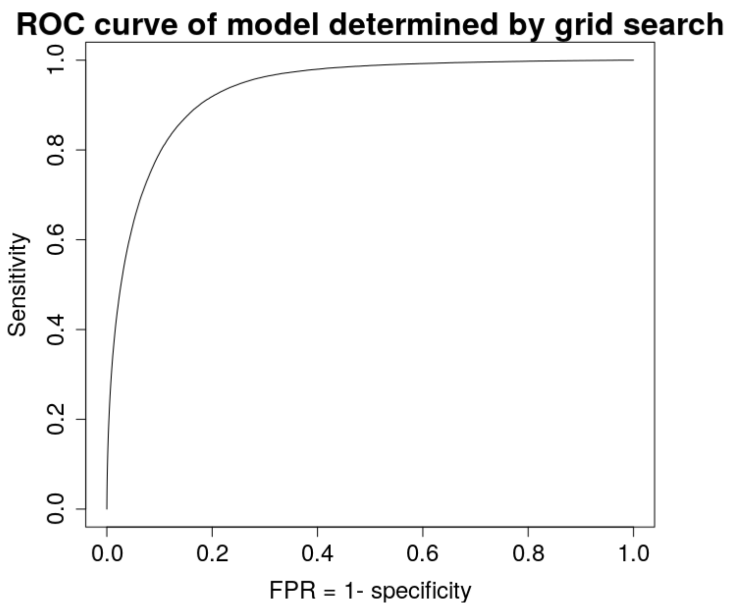


Figure 16: ROC curve of the locally optimal model as obtained by grid search. On the horizontal axis, the false positive rate is displayed, while sensitivity is mapped to the vertical axis. An ideal ROC curve would rise vertically until reaching a sensitivity of 1, then stretch horizontally.

5.2.2 Results of Random Search

Random search identifies the locally optimal model in a similar area of the searched sub-space of Θ, albeit after only 266 trials. Here, the best hyper-parameter combination consists of

- 20 factors,
- confidence parameter $\alpha = 91.94$, and
- regularization parameter $\lambda = 24.42$.

Like grid search, 20 factors are deemed locally optimal. Also, confidence parameter α remains in the neighborhood of the value grid search found to work best; regularization parameter λ is somewhat higher though. Model performance is slightly below that of the one obtained by grid search and is assigned an average rank of 7.27%, so items actually purchased in the test phase are on average located within the top 7.27% of the recommendation list. The locally optimal model is situated just above the foremost bottom corner of the cubical sub-space of Θ displayed in Figure 17. In contrast to the even structure observed in grid search, trial points are now scattered throughout the cube.

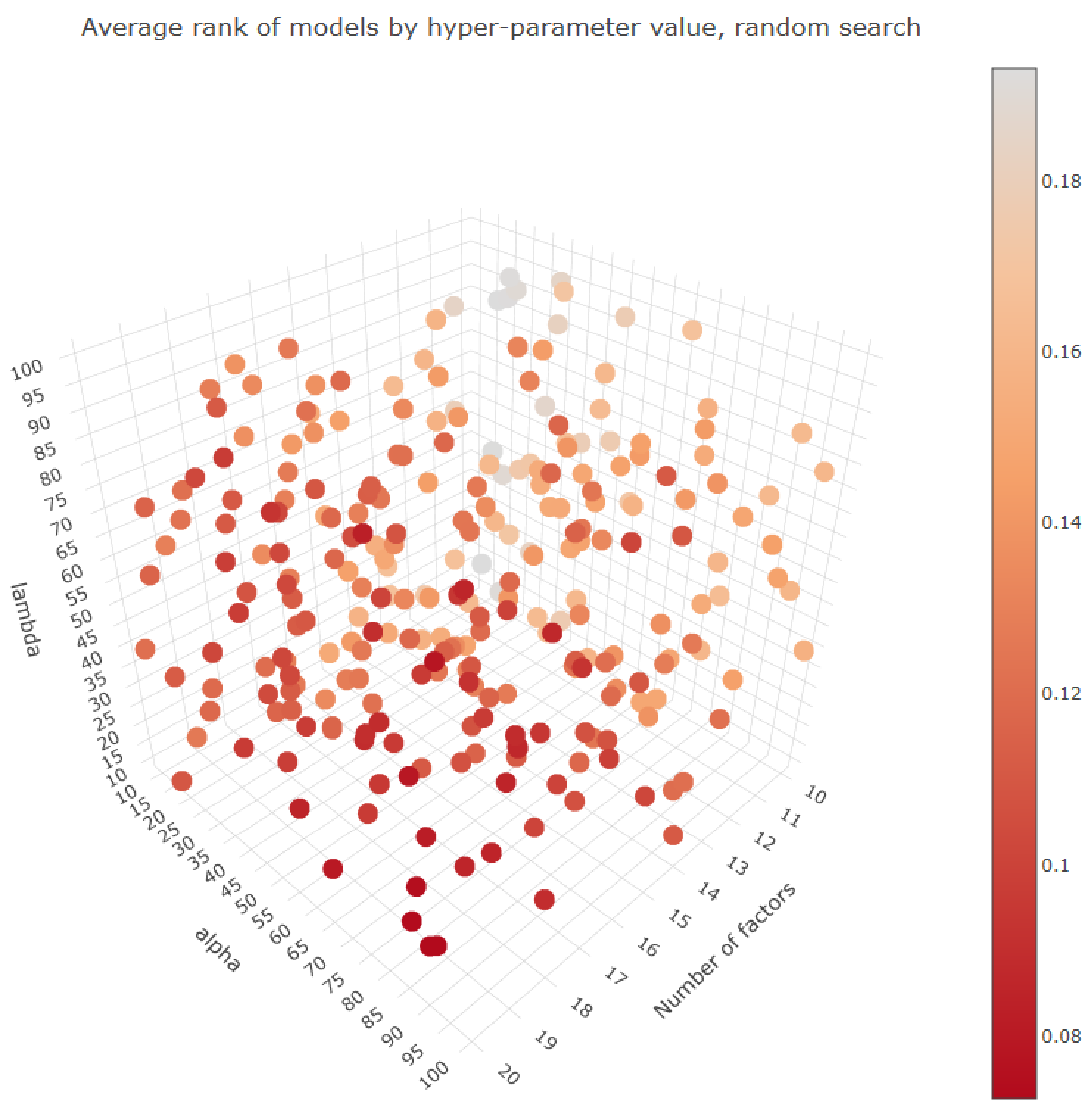


Figure 17: Average rank of models by hyper-parameter value as obtained by random search. The three hyper-parameters factor amount, α and λ are mapped to the x-axis, y-axis and z-axis, respectively. Each point in the thus constituted space represents a different model with corresponding hyper-parameter values. Darker colors indicate lower average rank values and therefore better models.

The ROC curve for the model selected in random search, depicted in Figure 18, appears almost identical to the one in grid search. Consequently, its AUC is almost as high with 92.88%. Both approaches are thus very well capable of discriminating between relevant and irrelevant items.

While this may seem desirable at first glance, the high number of factors chosen to fit the two models and the good recommendation results give rise to the concern that both might be *over-specialized*. This problem will be discussed in the following Section.

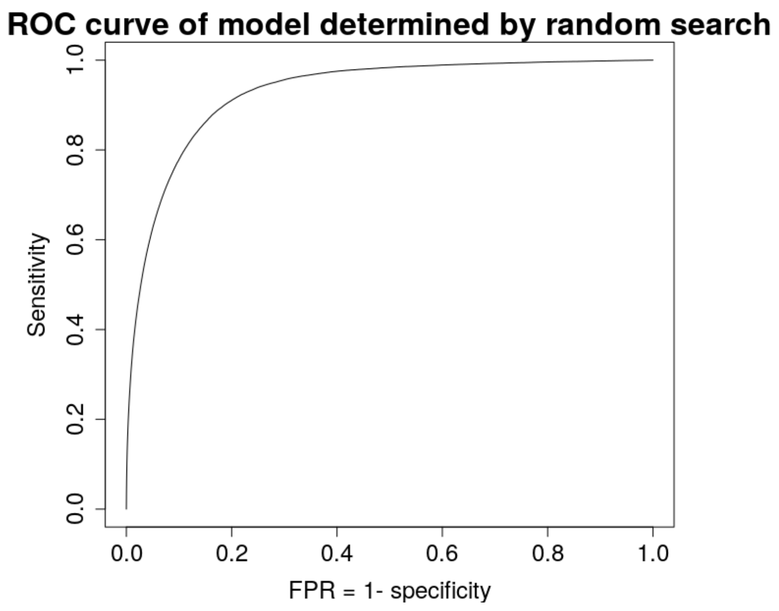


Figure 18: ROC curve of the locally optimal model as obtained by random search. On the horizontal axis, the false positive rate is displayed, while sensitivity is mapped to the vertical axis. An ideal ROC curve would rise vertically until reaching a sensitivity of 1, then stretch horizontally.

5.3 Discussion

5.3.1 Critical Assessment of Both Approaches

The first conclusion to be drawn from the results discussed above is that random search for hyper-parameter optimization should be preferred over grid search. Although grid search finds a slightly better model, this advantage becomes insignificant when considering the fact that it took five-fold computation time for grid search to come up with this result. Time being costly, as argued earlier on, the slender difference in average rank values will hardly suffice for choosing grid search nonetheless. Other research offers similar results, suggesting that, despite its simplicity, random search is an efficient methodology for hyper-parameter optimization (at least if the number of hyper-parameters to be tuned is small). For instance, Mantovani et al. (2015) find that for optimization of support vector machines, random search performs equally well as grid search and also as more complex algorithms, but with much less computational effort (Mantovani et al., 2015). Bergstra and Bengio (2012) reach the same conclusion and even obtain better models through random search in some cases. They attribute the relative disadvantage of grid search mainly to the fact that it covers all dimensions of the search space evenly, even though model quality is often mainly driven by few hyper-parameters. Random search, by contrast, avoids allocating too much time for searching unpromising areas of Θ. Another positive aspect arising from randomness is that new trial points can be added at any time, where grid search would require to adjust the entire search grid at high computational costs (Bergstra and Bengio, 2012). So, in an environment where computation power is expensive, random search is shown to be an efficient approach to optimizing model hyper-parameters.

However, as has been hinted at in the previous section, the very good results of both approaches might actually point to a heavy over-specialization. This term refers to the tendency of collaborative filtering models to anchor recommendations at past user behavior overly strongly (see for example Adamopoulos and Tuzhilin (2014)). Such a propensity is to some degree inherent to collaborative filtering but often stands in conflict to beyond-accuracy recommendation goals. Narrowly specialized models such as the ones fitted here will struggle to offer, say, novel or broadly diversified items to users (Adamopoulos and Tuzhilin, 2014). Interestingly, the original work by Hu et al. (2008), whose implementation achieves an average rank of 8.35% and thus appears similarly biased towards accuracy, does not mention the issue. Within the scope of this Thesis and the data used herein, it can be concluded that the Hu-Koren-Volinsky model succeeds at capturing past user behavior quite well. An actual application must surely consider modifications so as to balance model accuracy with other goals of recommendation. In the last section, some conceivable enhancements to the Hu-Koren-Volinsky model as implemented here are discussed.

5.3.2 Possible Enhancements

Research on collaborative filtering for implicit feedback suggests there is still ample room for optimization. First and foremost, the model implemented in this analysis cannot handle large numbers of users and items well and thus offers only very limited scalability. This shortcoming, though major, can easily be fixed by building a more powerful computational infrastructure. There are other aspects, however, that lie beyond the scope of the Hu-Koren-Volinsky model as implemented here: improvements might be gained from including additional aspects of the data into the model, using alternative model families, or performing hyper-parameter tuning differently.

Modifying Data Input

While it is an inherent strength of collaborative filtering methods that they draw solely on historic user-item interactions, higher recommendation quality might be achieved from incorporating additional information, particularly so if such data is available without further costs. One aspect that is, for instance, discussed by Leimstoll and Stormer (2007) is *seasonality*. Seasonal items are on offer only for certain time periods. In order to avoid recommending to a user items that cannot even be purchased at the time and are thus temporarily irrelevant, the system should be able to identify all seasonal items along with their respective selling period and tweak recommendations accordingly. The former is a concern of product management, the latter requires alterations of the recommendation list (Leimstoll and Stormer, 2007).

Besides fluctuations that affect all users simultaneously, relevance of items might also vary on a user-individual level in a sense that it might be *time-dependent*. As Ding and Li (2005) argue, it is easily conceivable that since an interaction dating back several years a user's preferences may have changed substantially; recent purchases, by contrast, should give a more reliable idea on the current user profile. However, classic collaborative filtering methods and the Hu-Koren-Volinsky model in particular do not distinguish between old - possibly outdated - and recently observed data. In order to keep their recommendations relevant, they must be re-trained on a regular basis. This shortcoming can be mitigated in different ways. Obviously, older data could simply be omitted from analysis, but since observations in implicit feedback are typically very scarce already, this might aggravate the sparsity problem to an extent where meaningful recommendations are no longer possible (Ding and Li, 2005). Also, research suggests that older data contribute less but still positively to model quality (de Pessemier et al., 2010). As an alternative, several types of models that capture temporal dynamics have been developed. Vinagre et al. (2015) cluster these into two different approaches. *Time-aware* algorithms seek to exploit temporal patterns by enriching data input with time stamps. This way, the change of user behavior over the course of a day, week or year can be considered for recommendations. *Time-dependent methods* interpret data as a chronological sequence and strive to identify unprecedented behavioral changes rather than cyclical patterns. In both variations models can be technically constructed by projecting the user-item matrix into a temporal dimension, creating a three-dimensional tensor that is then factorized (Vinagre et al., 2015).

Another conceivable enhancement also tackles the convenient but potentially sub-optimal simplicity of purely transactional data input. Borrowing the idea behind content-based recommender systems, data is enriched by user and item *features* that are believed to carry valuable information not entirely captured by latent factors (or not extractable from factors by human operators). In what they have dubbed *attentive collaborative filtering*, Chen et al. (2017) attempt to make use of additional information from item features. They identify two forms of implicitness of feedback. The first concerns the selection of items itself in a sense that preferences cannot be directly derived from observing or not observing interactions. This dimension of implicitness is fully acknowledged and covered by the Hu-Koren-Volinsky model already. However, the authors point to a second dimension regarding the content of items: even in the event of a user preferring an item, it cannot be known whether all components of the item are approved of since there is typically no explicit feedback. They propose an integrated model capable of incorporating component-related information in the form of item features, which could well improve the implemented model also (Chen et al., 2017).

Choosing Alternative Models

Apart from data enrichment it is certainly worth considering to alter the type of model altogether. He et al. (2017) argue that the inner product of loadings vectors x_u and y_i, which is taken to predict user u's affinity towards item i (see Equation 14), fails to capture all the complex user-item interactions taking place in the low-dimensional factor space. In order to mitigate this shortcoming, models must be allowed to perform the representation with a large number of factors, which might in turn lead to over-specialization with respect to the training data. The fact that both grid and random search find the model with the maximum amount of factors to be locally optimal in this analysis might be related to this observation. The authors propose a model based on *neural networks* which they show to be a generalization of matrix factorization (He et al., 2017). A similar technique is examined by Covington et al. (2016) who apply a generalized matrix factorization framework in form of a pair of neural networks to video recommendations. Their studies also suggest that these models are capable of outperforming classic matrix factorization (Covington et al., 2016). Such approaches certainly appear promising, and the recency of those and other researchers' work in this area hint at the untapped potential that might still be hidden in the field of collaborative filtering.

Performing Hyper-Parameter Optimization Differently

Lastly, the process of hyper-parameter optimization on any model, potentially modified with some of the aforementioned proposals, could be carried out in a different way. To begin with, there are various alternative evaluation criteria applicable to the optimization process, as has been discussed in Section 2.3.4. In what has already been identified as a shortcoming of the modeling process implemented here, model quality is solely based on ranking accuracy. This introduces a natural bias towards more frequently purchased items. The operator of a recommender system, however, might want to take additional aspects, such as serendipity or item diversity, into account. For example, Wang et al. (2018) propose a model which incorporates item diversity, measured through the degree of orthogonality between item vectors, directly into the objective function. Vargas et al. (2011) define a two-step process where recommendation lists obtained through matrix factorization are manipulated with a diversification algorithm whose objective function seeks to balance similarity and diversity. An alternative approach, suggested by (Cheng et al., 2017), couples the two optimization problems in a supervised-learning setting and implements a single, learning-based collaborative filtering algorithm.

Besides modification of the evaluation criterion, model optimization might be ameliorated by using alternative tuning methods. Grid and random search are simple and easy to implement but perform optimization in a brute-force manner. In machine learning, hyper-parameter tuning via *Bayesian optimization* has been an object of active research. Rather than applying many different hyper-parameter values in a trial-and-error manner, the Bayesian approach aims at reducing the number of trials to a minimum. It views hyper-parameter tuning as a problem with an unknown objective function. In order to solve this problem, some surrogate model that incorporates prior belief and updates information throughout the process is used to estimate the objective function. A so-called acquisition function proposes a sampling point out of the search space Θ_{trial} which has a high probability of improving the model. The surrogate function is then evaluated at this point. With this new information, the updated acquisition function directs the search to the next-best hyper-parameter combination (Snoek et al., 2012). Since each trial of hyper-parameter combinations within latent factor models requires fresh factorization of typically very large matrices, speeding up the tuning process this way might be a another considerable improvement.

6 Conclusion

This Thesis has implemented a latent factor model, originally proposed by Hu et al. (2008), as a collaborative-filtering-based recommender system for German retail data. Latent factor models take a user-item ratings matrix, each of whose elements represents the rating of a specific user u on an item i, as input and performs factorization into two smaller matrices that contain users' and items' loadings on certain latent factors. Latent factors in this context are best thought of as user affinities that items satisfy to varying degree. This way, the typically high-dimensional data is compressed into a low-rank space in which both users and items are represented by the same dimensions. Crucially, factorization is successful even if the ratings matrix has missing values. These unobserved ratings can be estimated by taking the cross-product of the respective user and item loadings vectors. However, the implicit nature of the user feedback processed in this model, namely binary transaction data, impedes the identification of user preferences in a sense that unobserved transactions do by no means allow to immediately conclude a lack of preference. Rather, observed and unobserved transactions are treated with different levels of confidence. In order to find the optimal latent-factor representation of the user-item correlation structure, the model minimizes the confidence-weighted sum of prediction errors. The model was evaluated by comparing recommendations with real transactions of a later time period. Ranking accuracy served as an evaluation criterion: ideally, all items actually purchased in the test phase should be located high up the recommendation list. The three hyper-parameters shaping the Hu-Koren-Volinsky model - the number of latent factors, the difference in confidence levels of observed and unobserved ratings, and the regularization parameter - were optimized using grid search and random search as contesting tuning techniques. Both methods found quite similar results, but random search did so much more efficiently, needing only 20% of the number of trials in grid search to achieve roughly equivalent quality. Indeed, the locally optimal models constructed by both approaches achieved a very good ranking accuracy. Yet this highly accurate recreation of past user behavior stands in conflict with other objectives of recommendations, such as proposing novel or diverse items, and fails to take changes in behavior into account. Therefore, while the Hu-Koren-Volinsky model has proven to derive accurate recommendations for the data analyzed herein, several fields of potential improvement have been identified. These include, among others, boosting scalability and the incorporation of beyond-accuracy recommendation goals. The task of recommendation in general remains a topic attracting plenty of research. Certainly, the future will yet see a lot of fresh ideas and progress on the issue.

A Appendix

A.1 First-Order Conditions in the Hu-Koren-Volinsky Model

While the original paper by Hu et al. (2008) does not provide step-wise calculations, a development of the first-order conditions of the author's own account may be found here. Since conditions for x_u and y_i can be obtained in analogous manner, only calculations for user u are listed.

The objective function of the Hu-Koren-Volinsky model for user u is given by

$$\begin{aligned}\underset{X,Y}{\arg\min}\, J_{HKV,u} &= \underset{X,Y}{\arg\min} \sum_{i=1}^{n} c_{ui}\left(p_{ui} - x_u^T y_i\right)^2 + \lambda\left(\|x_u\|_F^2 + \sum_{i=1}^{n}\|y_i\|_F^2\right)\\ &= \underset{X,Y}{\arg\min}\, C^u\left(p_u - x_u^T Y\right)^2 + \lambda\left(\|x_u\|_F^2 + \|Y\|_F^2\right),\end{aligned}$$

which, by differentiation for x_u, leads to the first-order condition of Equation 17:

$$\begin{aligned} & -2Y^T C^u\left(p_u - x_u^T Y\right) + 2\lambda x_u = 0\\ \Leftrightarrow\quad & Y^T C^u\left(p_u - x_u^T Y\right) - \lambda x_u = 0\\ \Leftrightarrow\quad & Y^T C^u p_u - Y^T C^u x_u^T Y - \lambda x_u = 0\\ \Leftrightarrow\quad & Y^T C^u p_u = Y^T C^u x_u^T Y + \lambda x_u\\ \Leftrightarrow\quad & Y^T C^u p_u = x_u\left(Y^T C^u Y + \lambda I\right)\\ \Leftrightarrow\quad & x_u = \left(Y^T C^u Y + \lambda I\right)^{-1} Y^T C^u p_u\end{aligned}$$

A.2 SQL Query for Data Extraction

For the sake of anonymity, names of tables and columns have been modified so as not to be recognizable for anyone who might know the company the data originates from. In fact, the query is executed twice with different ⟨start⟩ and ⟨end⟩ parameters in order to obtain training and test data. The extraction of test data is thereby restricted to the user and item IDs of the training data.

```
SELECT a.uid
, a.iid
FROM salestable AS a

INNER JOIN (

    SELECT DISTINCT aa.uid
    , COUNT(*) AS npurchased
    FROM salestable AS aa

    INNER JOIN (

        SELECT DISTINCT uid
        , MIN(fromdate) AS fromdate
        , MAX(tilldate) as tilldate
        FROM enrolmenttable
        WHERE country = 'GERMANY'
        GROUP BY 1

        ) bb    ON aa.uid = bb.uid
                AND fromdate <= '2018-09-01'
                AND tilldate >= '2019-08-31'

    INNER JOIN (

        SELECT uid
        FROM (

            SELECT DISTINCT uid
            , COUNT(*) AS npurchased
            FROM salestable
            WHERE   1=1
                    AND date BETWEEN '2018-09-01'
                        AND '2019-08-31'
                    AND country = 'GERMANY'
            GROUP BY 1
            HAVING npurchased <= 50

            ) aaa
        ) cc    ON aa.uid = cc.uid

    WHERE   1=1
            AND date BETWEEN '2018-09-01'
                AND '2019-08-31'
            AND transactiontype = 'PURCHASE'
    GROUP BY 1
    QUALIFY ROW_NUMBER() OVER
        (ORDER BY npurchased DESC) <= 10000

) b ON a.uid = b.uid

INNER JOIN (

    SELECT DISTINCT aa.iid
    , COUNT(*) AS npurchased
    FROM salestable AS aa
    INNER JOIN (

        SELECT DISTINCT iid
        , MIN(releasedate) AS fromdate
        FROM producttable
        WHERE isdeleted = 0
        GROUP BY 1

        ) bb    ON aa.iid = bb.iid
                AND fromdate <= '2018-09-01'

    LEFT JOIN producttable cc
        ON aa.iid = cc.iid

    WHERE   1=1
            AND date BETWEEN '2018-09-01'
                AND '2019-08-31'
            AND transactiontype = 'PURCHASE'
            AND category = 'APPLICABLE'
    GROUP BY 1
    QUALIFY ROW_NUMBER() OVER
        (ORDER BY npurchased DESC) <= 3000

) c ON a.iid = c.iid

WHERE date BETWEEN <start> AND <end>
```

B Electronic Appendix

This Thesis is supplemented by additional material submitted in electronic form.

References

Adamopoulos, P. and Tuzhilin, A. (2014). On over-specialization and concentration bias of recommendations: Probabilistic neighborhood selection in collaborative filtering systems, *8th ACM Conference on Recommender Systems*, Foster City, California, USA.

Adomavicius, G. and Tuzhilin, A. (2011). Context-aware recommender systems, *in* F. Ricci, L. Rokach, B. Shapira and P. B. Kantor (eds), *Recommender Systems Handbook*, Springer.

Aggarwal, C. C. (2016). *Recommender Systems. The Textbook*, Springer.

Anandkumar, A. and Ge, R. (2016). Efficient approaches for escaping higher order saddle points in non-convex optimization, *JMLR Workshop and Conference Proceedings*, New York, New York, USA.

Bates, D., Maechler, M. and Davis, T. A. (2019). *Matrix: Sparse and Dense Matrix Classes and Methods*. R package version 1.2-17.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization, *Journal of Machine Learning Research* **13**: 281–305.

Bhojanapalli, S. and Neyshabur, B. (2016). Global optimality of local search for low rank matrix recovery, *30th Conference on Neural Information Processing Systems*, Barcelona, Spain.

Bobadilla, J., Ortega, F., Hernando, A. and Gutiérrez, A. (2013). Recommender systems survey, *Knowledge-Based Systems* **46**: 109–132.

Burke, R. (2000). Knowledge-based recommender systems, *Encyclopedia of Library and Information Systems* **69**.

Chen, J., Zhang, H., He, X., Nie, L., Liu, W. and Chua, T.-S. (2017). Attentive collaborative filtering: Multimedia recommendation with item- and component-level attention, *Proceedings of SIGIR'17*, At Kona, Hawaii, USA.

Cheng, P., Wang, S., Ma, J., Sun, J. and Xiong, H. (2017). Learning to recommend accurate and diverse items, *International World Wide Web Conference Committee (IW3C2)*, Perth, Australia.

Cheung, D. W., Ng, V. T., Fu, A. W. and Fu, Y. (1996). Efficient mining of association rules in distributed databases, *IEEE Transactions On Knowledge And Data Engineering* **8**(6): 911–922.

Covington, P., Adams, J. and Sargin, E. (2016). Deep neural networks for youtube recommendations, *10th ACM Conference on Recommender Systems*, Boston, Massachussets, USA.

de Pessemier, T., Dooms, S., Deryker, T. and Martens, L. (2010). Time dependency of data quality for collaborative filtering algorithms, *Proceedings of the fourth ACM conference on Recommender systems*, Barcelona, Spain.

Deelman, E., Singh, G., Livny, M., Berriman, B. and Good, J. (2008). The cost of doing science on the cloud: The montage example, *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, Shinjuku, Tokyo, Japan.

Ding, Y. and Li, X. (2005). Time weight collaborative filtering, *CIKM'05*, Bremen, Germany.

Ekstrand, M. D., Riedl, J. T. and Konstan, J. A. (2010). Collaborative filtering recommender systems, *Foundations and Trends in Human Computer Interaction* **4**(2): 81–173.

Fu, W., Nair, V. and Menzies, T. (2016). Why is differential evolution better than grid search for tuning defect predictors?, *arXiv e-prints* .

Goldberg, D., Nichols, D., Oki, B. and Terry, D. (1992). Using collaborative filtering to weave an information tapestry, *Communications of the ACM - Special issue on information filtering* **35**(12): 61–70.

Gouvert, O., Oberlin, T. and Févotte, C. (2018). Matrix co-factorization for cold-start recommendation, *Proceedings of the 13th International Conference on Web Information Systems and Technologies (WEBIST)*, Porto, Portugal.

Gras, B., Brun, A. and Boyer, A. (2017). Can matrix factorization improve the accuracy of recommendations provided to grey sheep users?, *Proceedings of the 19th International Society for Music Information Retrieval Conference*, Paris, France.

He, X., Liao, L., Zhang, H., Nie, L., Hu, X. and Chua, T.-S. (2017). Neural collaborative filtering, *WWW 2017*, Perth, Australia.

He, Y., Wang, C. and Jiang, C. (2019). Correlated matrix factorization for recommendation with implicit feedback, *IEEE Transactions on Knowledge and Data Engineering* **31**(3): 451–464.

Herlocker, J. L., Konstan, J. A., Terveen, L. G. and Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems, *ACM Transactions on Information Systems* **22**(1): 5–53.

Hiriart-Urruty, J.-B. (1995). Conditions for global optimality, *in* R. Horst and P. M. Pardalos (eds), *Handbook of Global Optimization*, Springer Science+Business Media.

Hu, Y., Koren, Y. and Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets, *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008)*, Pisa, Italy.

Huang, Z., Chen, H. and Zeng, D. (2004). Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering, *ACM Transactions on Information Systems* **22**(1): 1–104.

Hurley, N. and Zhang, M. (2011). Novelty and diversity in top-n recommendation – analysis and evaluation, *ACM Transactions on Internet Technology* **10**(4): 116–142.

Häubl, G. and Trifts, V. (2000). Consumer decision making in online shopping environments: The effects of interactive decision aids, *Marketing Science* **19**(1): 14:1–14:30.

Jain, P. and Kar, P. (2017). Non-convex optimization for machine learning, *Foundations and Trends in Machine Learning* **10**(3-4): 142–336.

Jawaheer, G., Szomszor, M. and Kostkova, P. (2010). Comparison of implicit and explicit feedback from an online music recommendation service, *Proceedings of the 1st International Workshop on Information Heterogeneity and Fusion in Recommender Systems*, Barcelona, Spain.

Karydi, E. and Margaritis, K. (2016). Parallel and distributed collaborative filtering: A survey, *ACM Computing Surveys (CSUR)* **49**(2): article no. 37.

Kelley, C. (1999). *Iterative Methods for Optimization*, Society for Industrial and Applied Mathematics.

Khusro, S., Ali, Z. and Ullah, I. (2016). Recommender systems: Issues, challenges, and research opportunities, *in* K. J. Kim and N. Joukov (eds), *Information Science and Applications (ICISA) 2016. Lecture Notes in Electrical Engineering*, Springer.

Klatzer, T. and Pock, T. (2015). Continuous hyper-parameter learning for support vector machines, *Proceedings of the 20th Computer Vision Winter Workshop*, Seggau, Austria.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection, *IJCAI 95 Proceedings of the 14th international joint conference on Artificial intelligence*, Montreal, Quebec, Canada.

Koren, Y. (2008). Factorization meets the neighborhood: a multifaceted collaborative filtering model, *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, Las Vegas, Nevada, USA.

Kotkov, D., Wang, S. and Veijalainen, J. (2016). A survey of serendipity in recommender systems, *Knowledge-Based Systems* **111**(C): 180–192.

Krone, J., Ogden, W. F. and Sitaraman, M. (2003). Oo big o: A sensitive notation for software engineering, *Technical report*, Clemson University, Department of Computer Science.

Leimstoll, U. and Stormer, H. (2007). Collaborative filtering recommender systems for online shops, *Proceedings of the Thirteenth Americas Conference on Information Systems*, Keystone, Colorado, USA.

Liu, H. and Cocea, M. (2017). Semi-random partitioning of data into training and test sets in granular computing context, *Granular Computing* **2**(4): 357–386.

Loehlin, J. C. and Beaujean, A. A. (2017). *Latent Variable Models. An Introduction to Factor, Path, and Structural Equation Analysis*, 5 edn, Routledge.

Luo, G. (2016). A review of automatic selection methods for machine learning algorithms and hyperparameter values, *Network Modeling Analysis in Health Informatics and Bioinformatics* **5**: 1–16.

Lü, L., Medo, M., Yeung, C. H., Zhang, Y.-C., Zhang, Z.-K. and Zhou, T. (2012). Recommender systems, *Physics Reports* **519**(1): 1–49.

Mantovani, R. G., Rossi, A. L. D., Vanschoren, J., Bischl, B. and de Carvalho, A. C. P. L. F. (2015). Effectiveness of random search in svm hyper-parameter tuning, *2015 International Joint Conference on Neural Networks (IJCNN)*, Killarney, Ireland.

Melville, P. and Sindhwani, V. (2017). Recommender systems, *in* C. Sammut and G. Webb (eds), *Encyclopedia of Machine Learning and Data Mining*, Springer.

Nocedal, J. and Wright, S. J. (1999). *Numerical Optimization*, Springer.

Ooi, H. and Weston, S. (2019). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics.* R package version 1.4.7.

Ooi, H., Weston, S. and Tenenbaum, D. (2019). *doParallel: Foreach Parallel Adaptor for the 'parallel' Package.* R package version 1.0.15.

Pan, R., Zhou, Y., Cao, B., Liu, N. N., Lukose, R., Scholz, M. and Yang, Q. (2008). One-class collaborative filtering, *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, Antwerp, Belgium.

Papadakis, M., Tsagris, M., Dimitriadis, M., Fafalios, S., Tsamardinos, I., Fasiolo, M., Borboudakis, G., Burkardt, J., Zou, C., Lakiotaki, K. and Chatzipantsiou, C. (2019). *Rfast: A Collection of Efficient and Extremely Fast R Functions.* R package version 1.9.5.

R Core Team (2019). *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria.

Ricci, F., Rokach, L. and Shapira, B. (2011). Introduction to recommender systems handbook, *in* F. Ricci, L. Rokach, B. Shapira and P. B. Kantor (eds), *Recommender Systems Handbook*, Springer.

Sarwar, B., Karypis, G., Konstan, J. and Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms, *Proceedings of the 10th international conference on World Wide Web*, Hong Kong.

Schafer, J. B., Konstan, J. and Riedl, J. (1999). Recommender systems in e-commerce, *Proceedings of the 1st ACM conference on Electronic commerce*, Denver, Colorado, USA.

Seroussi, Y., Bohnert, F. and Zukerman, I. (2011). Personalised rating prediction for new users using latent factor models, *HT '11 Proceedings of the 22nd ACM conference on Hypertext and hypermedia*, Eindhove, Netherlands.

Shani, G. and Gunawardana, A. (2011). Evaluating recommendation systems, *in* F. Ricci, L. Rokach, B. Shapira and P. B. Kantor (eds), *Recommender Systems Handbook*, Springer.

Sievert, C., Parmer, C., Hocking, T., Chamberlain, S., Ram, K., Corvellec, M. and Despouy, P. (2019). *Matrix: Sparse and Dense Matrix Classes and Methods*. R package version 4.9.0.

Sing, T., Sander, O., Beerenwinkel, N. and Lengauer, T. (2015). *ROCR: Visualizing the Performance of Scoring Classifiers*. R package version 1.0-7.

Sivapalan, S., Sadeghian, A. and Rahnama, H. (2014). Recommender systems in e-commerce, *World Automation Congress (WAC)*, At Kona, Hawaii, USA.

Snoek, J., Larochelle, H. and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms, *NIPS'12 Proceedings of the 25th International Conference on Neural Information Processing Systems*, Lake Tahoe, Nevada, USA.

Su, X. and Koshgoftaar, T. M. (2009). A survey of collaborative filtering techniques, *Advances in Artificial Intelligence* **22009**(4): 19 pages.

Surjanovic, S. and Bingham, D. (2017 (accessed on 2019/06/08)). Virtual library of simulation experiments: Test functions and datasets, `https://www.sfu.ca/~ssurjano.html`.

Takács, G., Pilászy, I., Németh, B. and Tikk, D. (2009). Scalable collaborative filtering approaches for large recommender systems, *Journal of Machine Learning Research* **10**: 623–656.

Vargas, S., Castells, P. and Vallet, D. (2011). Intent-oriented diversity in recommender systems, *SIGIR '11 Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, Beijing, China.

Victor, P., de Cock, M. and Cornelis, C. (2011). Trust and recommendations, *in* F. Ricci, L. Rokach, B. Shapira and P. B. Kantor (eds), *Recommender Systems Handbook*, Springer.

Vinagre, J., Jorge, A. M. and Gama, J. (2015). An overview on the exploitation of time in collaborative filtering, *WIREs Data Mining and Knowledge Discovery* **5**: 195–215.

Wang, J., Tian, F., Yu, H., Liu, C. H., Zhan, K. and Wang, X. (2018). Diverse non-negative matrix factorization for multiview data representation, *IEEE Transactions on Cybernetics* **48**: 2620–2632.

Wickham, H., Chang, W., Henry, L., Pedersen, T. L., Takahashi, K., Wilke, C., Woo, K. and Yutani, H. (2019). *foreach: Provides Foreach Looping Construct*. R package version 3.2.1.

Wickham, H., François, R., Henry, L. and Müller, K. (2019). *dplyr: A Grammar of Data Manipulation*. R package version 0.8.3.

Wickham, H. and Henry, L. (2019). *tidyr: Tidy Messy Data*. R package version 1.0.0.

Young, S. R., Rose, D. C., Karnowski, T. P., Lim, S.-H. and Patton, R. M. (2015). Optimizing deep learning hyper-parameters through an evolutionary algorithm, *MLHPC 15 Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, Austin, Texas, USA.

Yu, H.-F., Hsieh, C.-J., Si, S. and Dhillon, I. (2014). Parallel matrix factorization for recommender systems, *Knowledge and Information Systems* **41**(3): 793–819.

Zheng, A. (2015 (accessed on 2019/27/08)). Evaluating machine learning models, `https://www.oreilly.com/ideas/evaluating-machine-learning-models/page/5/hyperparameter-tuning`.

Declaration of Authorship

I hereby declare that the Thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references. I am aware that the Thesis in digital form can be examined for the use of unauthorized aid and in order to determine whether the Thesis as a whole or parts incorporated in it may be deemed as plagiarism. For the comparison of my work with existing sources I agree that it shall be entered in a database where it shall also remain after examination, to enable comparison with future Theses submitted. Further rights of reproduction and usage, however, are not granted here. This paper was not previously presented to another examination board and has not been published.