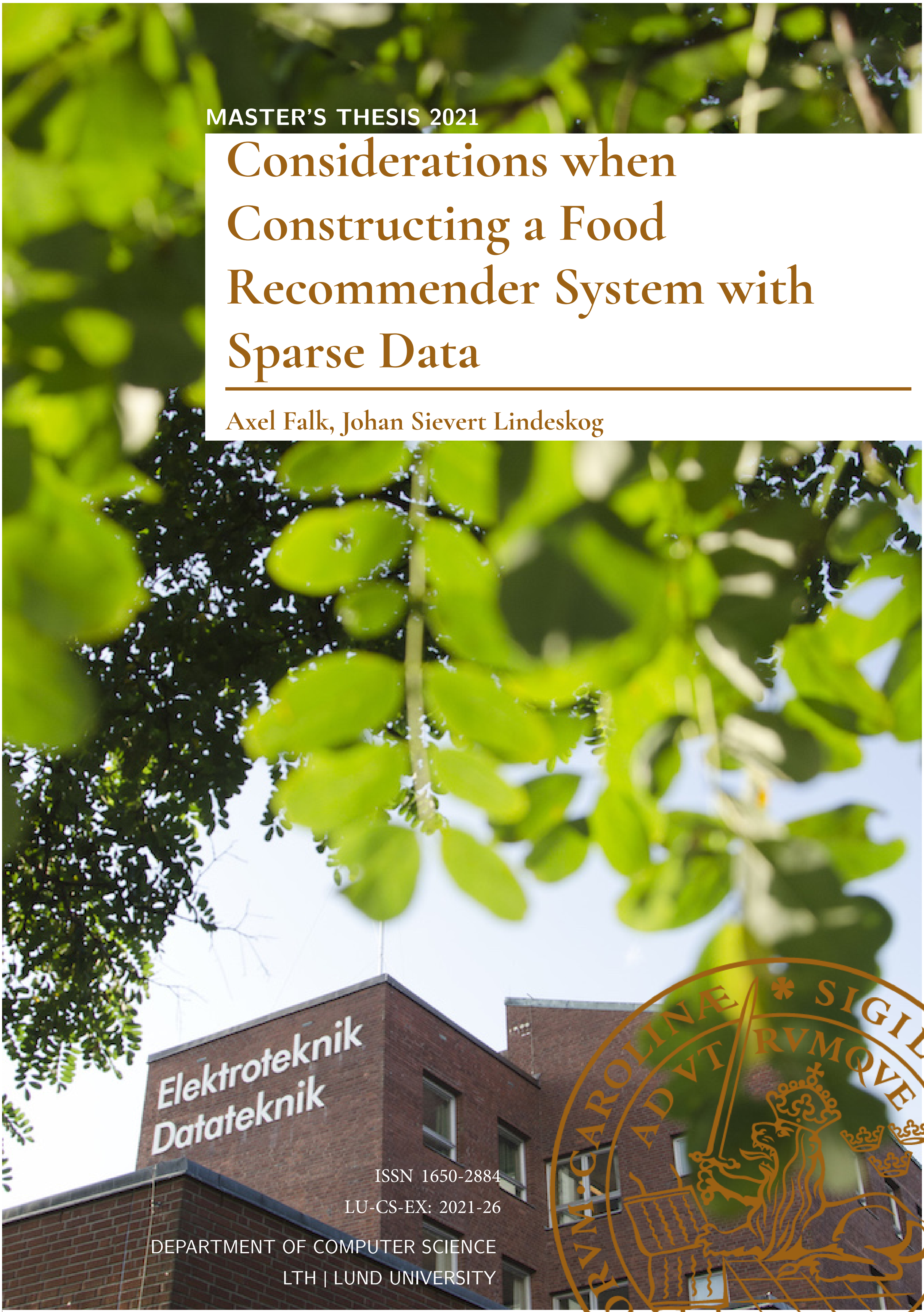
MASTER'S THESIS 2021

Considerations when Constructing a Food Recommender System with Sparse Data

Axel Falk, Johan Sievert Lindeskog



ISSN 1650-2884
LU-CS-EX: 2021-26

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2021-26

Considerations when Constructing a Food Recommender System with Sparse Data

Att tänka på när man implementerar ett rekommendationssystem baserat på gles data

Axel Falk, Johan Sievert Lindeskog

Considerations when Constructing a Food Recommender System with Sparse Data

Axel Falk
elt15afa@student.lu.se

Johan Sievert Lindeskog
johan.sievertlindeskog@gmail.com

July 6, 2021

Master's thesis work carried out at MEIQ Systems AB.

Supervisor: Dennis Medved (LTH), dennis.medved@med.lu.se
Jakob Navrozidis (WEIQ), jakob.navrozidis@weiq.tech

Examiner: Elin Anna Topp, elin_anna.topp@cs.lth.se

Abstract

In fast growing economies, companies connected to e-commerce rely on a recommender system, as these systems can improve the user experience and increase sales at the same time. In this thesis, we implement and investigate the performance of a food recommender system, based on implicit feedback data collected from over 100 restaurants in Sweden by MEIQ Systems AB. First, an attempt to convert the implicit feedback to pseudo explicit rating is investigated. Second, the implicit feedback is used without a conversion. Content-based filtering and collaborative filtering are the two main strategies that will be compared. Techniques like singular value decomposition and alternating least squares will be used. Our results show that the implicit interpretation of the data is preferable.

We will focus on the meaning of "good". What is a good recommendation? As it is not obvious how one can evaluate the results, we will discuss trade-offs, limitations and advantages, and ultimately provide suggestions and considerations for dealing with recommender systems of similar nature, with a focus on top-k recommendation evaluation.

Keywords: recommender system, collaborative filtering, matrix factorization, content-based filtering, item features

Acknowledgements

First of all, we would like to thank MEIQ Systems AB for providing the data set. Without the data, this thesis would not have been possible. Second, a big thank you to our supervisors Dennis Medved, postdoctoral researcher at the department of Computer Science at LTH, and Jakob Navrozidis, lead developer at MEIQ. Also, thank you Elin Anna Topp, the examiner of this thesis, for providing the final valuable insights and comments, making it possible for us to pass and thereby earn the honor of graduating as engineers. Last but not least, Isak Falk, Axel's brother and doctoral student at the University College London, has been yet another source of inspiration during this project. Thank you Isak for cheering on us.

Axel Falk & Johan Sievert Lindeskog, June 2021

Contents

Acronyms

ALS Alternating least squares.

CBF Content-based filtering.

CF Collaborative filtering.

MF Matrix factorization.

ML Machine learning.

NLP Natural language processing.

RMSE Root mean squared error.

RS Recommender system.

SVD Singular value decomposition.

TFIDF Term frequency–inverse document frequency.

WARP Weighted approximate-rank pairwise.

Nomenclature

$\hat{r}_{ui}$	the estimated rating of user u for item i
μ	the mean of all ratings
b_i	item bias, i.e., the mean of all ratings given to item i
b_{ui}	the baseline rating of user u for item i
b_u	user bias, i.e., the mean of all ratings given by user u
I	the set of all items. i and j denote items
U	the set of all users. u and v denote users

List of Figures

List of Tables

Chapter 1

Introduction

1.1 MEIQ Systems AB and WEIQ

This degree project is done in collaboration with MEIQ Systems AB (MEIQ), a start-up software company since 2018 based in Malmö, Sweden. The company conducts development, operation, and sales of a platform for menu and order management, aimed at the restaurant industry and compatible operations. Their main product is called "WEIQ" (a play on words of the question "Why queue?"), which is a free software application (app) available for anyone who wants to order food and drinks. The aim of WEIQ is primarily to remove queuing time for the customers as they place orders directly in the app, without involvement of the restaurant staff, and choose a time to pick up their order. This saves time, both for the customers and the restaurants, which is a crucial factor in the restaurant industry. Especially at crowded bars, where a fair and organized queuing system rarely exists, the app solution can be useful. Furthermore, the app minimizes the risk of incorrect orders as no involvement of waiters and waitresses is needed, and there is no way to dine and dash since the payment is required before the order is complete.

1.2 Historical background of recommender systems

The first form of a recommender system (RS) was presented in 1992 [18], used to filter spam from important emails. Since then, numerous e-commerce websites implemented these systems to make personalized recommendations to their customers [51]. In 2013, about 35% of what a consumer purchased at Amazon.com came from some form of product recommendation [36]. However, not only e-commerce websites use personalized recommendation. In 2006, Netflix announced that the winner of the *Netflix Prize*, an open competition for predicting user-item ratings on movies and series, would be awarded 1M dollars. This lead to a major interest within the field, with more than 20000 teams from 150 countries participating [4]. Today, 80% of Netflix watch time comes from recommendations [19]. Another big company where a recommendation engine is essential is YouTube, whose product chief claims that about 70% of the watch time on the platform comes from recommendations using artificial intelligence (AI) and machine learning (ML), rather than user search [55].

1.3 Related work

There are two types of input from which a RS is constructed; explicit feedback, where users explicitly express their preferences in some form, i.e., a rating scale, and implicit feedback, where users' interactions with products, i.e., clicking on an item or purchasing an item, are used to predict their preferences instead. Explicit feedback is preferable to work with as the feedback provided by the user is less ambiguous than its implicit counterpart [34]. With that being said, many systems are restricted to the usage of implicit feedback [26].

Today, most RS are either based on content-based filtering (CBF), recommending items similar to other items, or collaborative filtering (CF), recommending items based on user similarities. CF in particular has gained popularity over the years, originally by using nearest-neighbor techniques, but lately, with greater success, by the use of matrix factorization [11]. Hybrid models combine different filtering techniques, e.g., CBF and CF, in order to solve inherent problems of the particular filtering technique.

The trend in RS, similarly to many other fields, is currently deep learning [60]. While the deep learning based RS are state-of-the-art, they share reproducibility problems. Maurizio Ferrari Dacrema et al. published a paper in 2019 demonstrating that out of the 18 best RS algorithms from the previous year, only 7 of them were somewhat reproducible and out of those 7, only 1 could outperform typical heuristic methods such as nearest-neighbor based models [13].

1.4 Focus of this work

At the time of writing, WEIQ has no feature implemented to target the customers of the restaurants with tailored recommendations. A good RS can improve the user experience and increase sales at the same time, which is why an implementation of such a feature could be beneficial, both for the restaurants, the customers, and MEIQ as a company. Just like most software applications do, WEIQ collects data from its users. The objective from MEIQ's point of view is to, given that a user is browsing a particular restaurant's menu, display top 5 recommendations for the user, before they have decided what to order.

During this thesis, we will try to answer the following questions:

- Is it possible to implement a good RS based on implicit feedback using ML techniques, given that the data is incomplete, dirty, and very limited?
- How should the data be preprocessed?
- How should the models be evaluated and compared?
- Will CF be superior to CBF, despite the fact that the data is very limited?

As there already exist a lot of previous work in this field, we will focus on the meaning of good in relation to RS. As it is not obvious how one can evaluate the results, we will discuss trade-offs, limitations and advantages, and ultimately provide suggestions and considerations for dealing with RS of similar nature, with a focus on top-k recommendation evaluation.

1.4.1 Method

The objective of this thesis was to implement a RS based on the data given. We investigated the data characteristics to limit the scope of appropriate methods. As the data contained both user-item interactions and item features, we decided to try both CBF and CF. The CBF was implemented using the information retrieval method term frequency–inverse document frequency (TFIDF) together with the similarity measure cosine similarity, introduced in Section 2.2.3 and 2.2.4, respectively.

We considered two approaches in the case of CF, treating the implicit feedback as explicit, by a conversion, and using the original implicit feedback. Most research about RS concerns explicit feedback, which is why we tried the explicit approach. For this approach, we used the model-based CF method singular value decomposition (SVD), described in Section 2.4.2. For the second approach with implicit feedback, we used alternating least squares (ALS) and weighted approximate-rank pairwise (WARP) loss, explained in Section 2.4.3 and 2.4.4, respectively.

When evaluating the performance of the RS, the top recommendations were in focus. Therefore, we chose to use the classification accuracy metrics precision@k and recall@k, described in Section 2.6.2 and 2.6.3, respectively, when comparing the ALS-model and the WARP-model. These metrics could not be used directly when evaluating the SVD-model, which is why we here instead used the predictive accuracy metric root mean squared error (RMSE), explained in Section 2.6.1. Finally, we decided to compare the models subjectively by examining the recommendations given by each model for a specific user.

1.4.2 Contributions

During most of this thesis, we worked with the programming parts together in a pair programming fashion, although sub-tasks were delegated differently. In the later parts of the projects, Axel focused more on the CF parts containing WARP while Johan focused on the CBF parts. This included both implementation, modifications, and evaluation of the respective methods. Although the focus areas of this thesis were divided in the later parts, mostly due to limited amount of time, both writers of this thesis have been involved in some way during the whole project. The final last month were spent writing the thesis together, with support from notes and observations during the project.

1.4.3 Thesis outline

After introducing MEIQ as a company and presenting some historical background about RS, Chapter 2 includes essential theory about different types of filtering approaches and evaluation methods. Chapter 3 contains an overview of the data, and the general approach regarding the construction of the RS. The results are presented in Chapter 4 along with discussions. Finally, in Chapter 5, we finish the report with a conclusion, where we summarize and further discuss our findings, followed by potential areas of improvement.

Chapter 2
Theory and method

2.1 Data interpretation

To provide recommendations, other than random, some form of pattern must be extracted from the data. However, different types of feedback can be used in RS. Normally, this feedback is either explicit or implicit, which will be discussed below.

2.1.1 Explicit feedback

If a user u has the possibility of rating items they have interacted with, for instance, with a thumbs up/thumbs down rating, or on a scale from 1 to 5, this would be an example of explicit feedback. When u really likes an item i, this can be explicitly expressed with a high rating, i.e., we know that u likes i if the rating is high (given that u will never be biased and always give truthful ratings). The same applies for low ratings, i.e., u can explicitly express that i is not preferred by giving it a low rating.

Explicit feedback can be described in terms of an interaction matrix [15], where the rows and columns represent the users and items, respectively. An example of an interaction matrix, given the set of all users U, and the set of all items I, is displayed in Figure 2.1.

Here, the first user u_1 has interacted with items i_2, i_4, and i_5. Note that u_1 has never interacted with i_1 nor i_3, which is why the rating is set to 0. To be clear, this does not mean that u_1 dislikes i_1 or i_3, but rather that they have not interacted with the item.

2.1.2 Implicit feedback

In the case where no explicit information about user preferences is available, one must rely on other methods to understand the users' behavior. For instance, if a user purchases a product, without rating it, it is still considered as an interaction, although implicit. To give an example of an implicit interaction matrix, replace all numbers that are non-zero with 1 in Figure 2.1.

	Item 1	Item 2	Item 3	Item 4	Item 5
User 1	0	3	0	5	4
User 2	2	0	0	5	0
User 3	0	0	3	0	0
User 4	1	0	4	0	2
User 5	4	3	0	4	4

Figure 2.1: An example of a 5x5 explicit interaction matrix, displaying what items which user has interacted with, and which score the user rated the item with.

Browsing activity and watching habits, etc., could also be used to model user preferences [26]. Note that in this situation, we do not have any direct input regarding the preferences of the users, especially not about interactions with a negative outcome. If a user interacts with an item once, do they like it or not? How implicit feedback can be interpreted will be elaborated in Chapter 3.

Normally, explicit feedback is preferred over implicit [31], due to the fact that implicit feedback is much noisier and not as easy to work with. This should correspond to intuition, since, as mentioned before, we explicitly know the opinion regarding the items a user has interacted with in explicit feedback, compared to implicit feedback where we only know what items the user interacted with, not considering the opinion of the interaction. However, in practice, implicit data is more common to work with, due to the effective and easy way to collect it, for instance with cookies.

2.2 Natural language processing

Natural language processing (NLP) is a field within computer science that explores how computers can interact with natural languages, for instance, Swedish, Spanish and Chinese, i.e., languages that human societies use to communicate among themselves. The general problem with natural languages is that the grammatical and semantic (semantics, syntactic, and pragmatics) rules that each specific language has, are not necessarily followed by the speaker/writer of that language, but the context is often enough for the listener/reader to extract the information needed to comprehend the meaning. The intended goal with NLP in this project is to improve the consistency of the item features, described in Section 3.4.

2.2.1 Word embeddings

Word embedding or distributed word representation is a term used in NLP for vector representation of words. Words are mapped to a high dimensional vector space, and hopefully, similar words will be mapped close to each other in this vector space [39].

Since most restaurants use Swedish when naming tags and categories, a word embedding trained on a Swedish corpus is used. fastText is an open-source library created by Facebook's AI Research lab (FAIR) focused on text representation and text classification and provides pretrained word vector models for a multitude of languages, amongst them Swedish [8]. The word vectors were trained using an improved Continuous Bag of Words (CBOW) model on corpora gathered from Wikipedia [20]. Continuous Bag of Words is a word vector model that contextualizes words of a corpus and maps them to the vector-space [38].

2.2.2 Clustering methods

When the words are mapped to a vector space by the embeddings, there are mathematical methods to cluster the words. The resulting clusters will contain similar words. The clustering method used in the project was k-means clustering. While other methods were experimented with, such as Spectral clustering [57] and T-distributed Stochastic Neighbor Embedding [24], they performed worse and were thus excluded from the results.

The k-means clustering algorithm is initialized by randomizing the locations of the centroids in the vector space, followed by the assignment step where each observation is assigned a cluster based on the distance to the nearest centroid. This is followed by the update step, where the position of each centroid is re-estimated in the vector space based on the new mean location of all the observations belonging to that cluster. These two steps are reiterated until convergence is achieved.

2.2.3 Term frequency–inverse document frequency

Term frequency–inverse document frequency (TFIDF) is a common method within the field of NLP, used for understanding the importance of a word in a collection of documents or a corpus [47]. The first part, term-frequency, is calculated as

$$\mathrm{tf}(t,d) = \frac{f_{t,d}}{T_d} \tag{2.1}$$

where t is the term (word), d is the document, $f_{t,d}$ is the number of times the term t occurs in document d, and T_d is the total number of terms $t' \in d$. The last part exists since the size of different documents can vary, which would make only raw count of words unfair to compare between documents. Hence, the term-frequency tries to capture the importance of a word in a specific document.

The second part, inverse document frequency, is calculated as

$$\mathrm{idf}(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|} \tag{2.2}$$

where N is the total number of documents in the collection and $|\{d \in D : t \in d\}|$ is the number of documents in the collection where t appears. Thus, frequently occurring words that show up in most of the documents in the collection will be weighted down since the logarithm of something small will be close to zero or negative.

Breitinger et al. conducted a survey in 2015 [3], investigating research papers concerning RS. They found that 83% of text-based RS in digital libraries use TFIDF for information extraction.

2.2.4 Cosine similarity

Just like TFIDF, cosine similarity is a method within the field of NLP, commonly used to measure document similarity in text analysis [21]. If every document is represented as a vector with its corresponding words as dimensions, the similarity between two vectors (documents) **A** and **B**, can be found by simply calculating the cosine of the angle between them according to

$$\textit{cosine similarity} = \cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{||\mathbf{A}||||\mathbf{B}||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}, \tag{2.3}$$

where the second equality comes from the definition of scalar product in linear algebra. Note that documents will probably contain a lot of words, making these vector representations high dimensional [54].

2.3 Content-based filtering

In general, algorithms used in RS belong to two types of categories: content-based filtering (CBF) and collaborative filtering (CF) [28]. In addition to this, there are hybrid models, as will be discussed in Section 2.4.4. In this section, an overview of CBF will be presented.

2.3.1 Item-item

An intuitive implementation of a CBF RS would be, as the name indicates, to use the content to recommend items. In this type of implementation, the content is regarding the items. If some kind of information about the items is available, for instance, categories, tags or a description in text, it could be used as features for describing the items. Assuming that item features are defined, combined with a method for comparing the similarity of items, e.g., TFIDF followed by cosine similarity, recommendations can be made based on the similarity score of items. If user u previously has interacted with item i, and item j is similar to i, then j will be recommended to u. A descriptive image of this situation is represented in the right part of Figure 2.2.

2.3.2 User-user

This variant of CBF is in some sense quite similar to the previous item-item implementation. The difference lies in that there exists little or no content about the items, and instead, content about the users exists. If, for instance, demographic information, such as gender, age, country, etc., would be available, one could use this as features to identify similar users

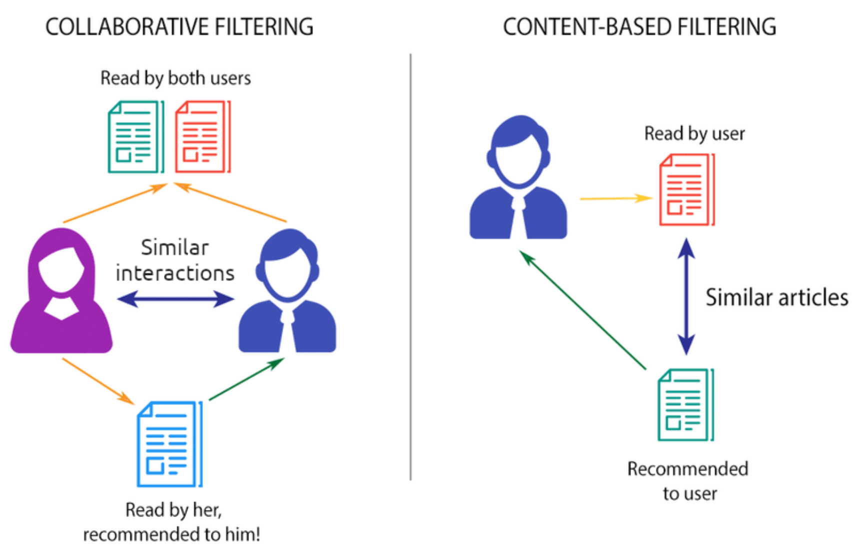


Figure 2.2: Figure of collaborative- (left) and content-based filtering (right). This is a modified version of an image from [25].

(although, for instance, women from Sweden in their 20's probably do not like the same food only because they share similar demographic information). From this, assuming that user u is similar to user v, one can recommend what u previously ordered to v, regardless of whether their purchase histories have something in common or not.

2.4 Collaborative filtering

Collaborative filtering is a technique used to recommend items to users based on the preferences of other users [50]. An example of this can be found on the left side of Figure 2.2, where both users have read the same articles and therefore are seen as having similar preferences, which in turn is why the article that has only been read by the left user is recommended to the right one. To give another example, looking at Figure 2.1, user u_1 and u_5 have similar ratings for item i_2, i_4, and i_5, making the users similar. In addition to this, u_5 gave i_1 a high rating, which is why i_1 should probably be recommended to u_1. This follows the presumption, on which CF is based, that if user u shares an opinion with user v, they are inclined to share other opinions as well.

The essential difference between CF and CBF, perhaps evident by now, is that CBF is limited by the content and its features when making recommendations, while CF makes recommendations based on the users' preferences and the preferences of similar users. However, CF does not naturally incorporate item features in its model as CBF does [1].

There are two major classes of CF, memory-based and model-based [10]. In memory-based CF, user-item ratings are stored in memory and used unaltered to find recommendations based on some similarity function [58], for example cosine similarity. Model-based CF develops a model, often by applying a machine learning approach beforehand that predicts user ratings [49]. There are different machine learning algorithms that can be applied to estimate the model, and we will focus on latent feature extraction which will be presented in the next section.

2.4.1 Matrix factorization

Matrix factorization (MF) subsumes a group of methods that factorize a matrix into products of smaller matrices, and are among the most used and effective methods of latent factor modeling [34]. Latent factor models are used to estimate ratings based on unknown low-dimensional representations, i.e., latent features [48]. For instance, one factor might represent popularity, another might represent salty food, and a third might explain if a person has children or not. Even though the last mentioned example might seem peculiar, it is important to note that the system itself cannot contextualize the patterns, it only finds those that seem to accurately represent the users. Therefore, ethical considerations may need to be accounted for [40], for instance, the model may find patterns based on ethnicity or sexuality, which could in turn enforce stereotypes.

2.4.2 Singular value decomposition

Singular value decomposition (SVD) is a type of MF that gained a lot of popularity during the Netflix prize. Simon Funk developed an approach to incorporate a SVD-based model to CF [16], as conventional SVD would result in unsatisfactory results. This is mainly due to the interaction matrix being sparse, and using only the few known entries would result in overfitting the model [32]. Figure 2.3 illustrates how a matrix A is factorized into two low rank matrices U_k and V_k, and a diagonal matrix Σ_k, where k is the rank of the matrix. It is important to note that the result will be an approximation of the original matrix A. The reconstructed matrix A_k, which is the product of U_k, Σ_k, and V_k, is obtained in a way such that it minimizes the Frobenius norm, which is equivalent to the L_2-norm, i.e., $||A - A_k||_2$.

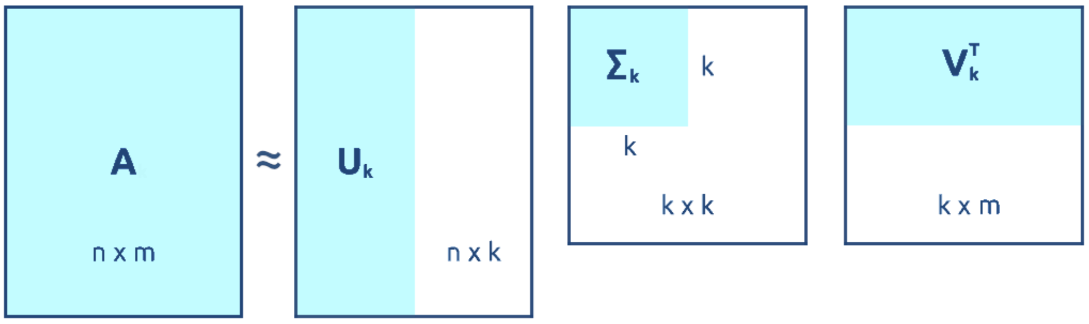


Figure 2.3: An example of a matrix A factorized into two low rank matrices U_k and V_k, and a diagonal matrix Σ_k, using SVD. Note that the resulting matrix is an approximation of the original matrix A.

The solution to the previous problem with the matrix being sparse is to fill the unknown entries with average ratings based on users and items, called user- and item bias, which would later be recalculated based on the SVD-algorithm. The predicted rating is estimated as

$$\hat{r}_{ui} = b_u + b_i + q_i^T p_u, \tag{2.4}$$

where b_u is the user bias, b_i the item bias, and q_i and p_u are the factor vectors for the items and users, respectively. The estimation of the parameters is done by minimizing the regularized squared error according to

$$\min_{p*,q*,b*} \sum_{(u,i)\in\kappa} (r_{ui} - b_u - b_i - p_u^T q_i)^2 + \lambda(||p_u||^2 + ||q_i||^2 + b_u^2 + b_i^2), \tag{2.5}$$

where κ is the set of user-item pairs for which r_{ui} is known. From Eq. 2.5, the parameters are tuned in the opposite direction of the gradient according to

$$b_u \longleftarrow b_u + \gamma(e_{ui} - \lambda b_u) \tag{2.6}$$

$$b_i \longleftarrow b_u + \gamma(e_{ui} - \lambda b_u) \tag{2.7}$$

$$p_u \longleftarrow p_u + \gamma(e_{ui} * q_i - \lambda p_u) \tag{2.8}$$

$$q_i \longleftarrow q_i + \gamma(e_{ui} * p_u - \lambda q_i) \tag{2.9}$$

where γ is the learning rate, λ the regularization term, and $e_{ui} = r_{ui} - \hat{r}_{ui}$. This procedure is repeated for every user-item pair, over a specified number of epochs.

2.4.3 Alternating least squares

Alternating least squares (ALS) is another method that belongs to the category of MF algorithms. This method is used both for explicit and implicit feedback [46, 26], while SVD is more common to use for explicit feedback. Both methods work well with sparse matrices, e.g., an interaction matrix. Broadly speaking, the algorithm iteratively tries to find the two low rank matrices U_k and V_k, as explained in Section 2.4.2, by first initializing them and then alternating between solving for the row factors (users in U_k), and column factors (items in V_k). This is done until some convergence is reached, typically within some tolerance. Ordinary least squares is used to solve for the row and column factors, respectively.

The details of the implementation can be found in [26], which will be used later to factorize an implicit interaction matrix.

2.4.4 Hybrid model- Weighted approximate-rank pairwise loss function

A hybrid model combines CBF and CF in the sense that it implements the latent factor model with item features. In this section, an implicit feedback hybrid model used in this project will be presented.

The role of loss functions is to measure an error between the algorithm's output, the predicted values, and the input, the true values, for the sake of optimizing the model. The error depends on the loss function, therefore in order to optimize the algorithm for a specific problem, a loss function that finds the appropriate error should be incorporated. A commonly used loss function is the squared distance, which sums up the squared errors between each true and predicted user-item pair [14] according to

$$L(y_{ui}, \hat{y}_{ui}) = (y_{ui} - \hat{y}_{ui})^2, \tag{2.10}$$

where y_{ui} is the true interaction score and $\hat{y}_{ui}$ is the predicted interaction score for an observed user-item pair.

Weighted approximate-rank pairwise (WARP) loss is a loss function that measures the precision@k, explained in Section 2.6.2, [59]. The ranking of labels i $\in \mathcal{Y}$ of an example u is the recurrent objective. For the labeled pairs (u, y), a randomly sampled correct y_i is considered correct.

WARP loss will randomly sample output labeled pairs (u, y) of a model for a specific u, until it finds a pair that it knows are wrongly labeled, and will then apply an update to these two incorrectly labeled examples. This is done in the following way:
For user u we have a target vector y, which contains the Boolean representation for all the items determined by user-item-interactions, i.e., 1's for interactions and 0's for non-interactions. Every item has a ranking score provided by the model. The score of the true user-item-interaction $(u, \hat{y}_i)$ is compared to the score of a randomly sampled user-item-interaction $(u, \hat{y}_{j \neq i})$. If $\hat{y}_i \geq \hat{y}_j$ then the score of a new random sample is compared to the score of the true sample. This is done until the occurrence where a random sample has a score greater than the true sample or the entirety of the vector has been iterated, which in the second case results in an accurate ranking prediction of the model. In the first case the loop is interrupted, and the error is

$$error(rank_i) = \hat{y}_j - \hat{y}_i. \tag{2.11}$$

A second step of the loss function is to include a factor which determined how well the ranking of the true label was in some regard to the entire set of items, and that is

$$\mathcal{L} = ln(\frac{X-1}{N})(\hat{y}_j - \hat{y}_i), \tag{2.12}$$

where X is the number of items and N is the number of randomly sampled items. This is done for every labeled pair in the data set. How this loss function is then implemented to find the stochastic gradient descent of the training of the model is somewhat more complicated and is therefore excluded from the report, but can be found in [59].

2.4.5 Other methods

Other methods were tried out in this project that have not been presented yet. For instance, non-negative matrix factorization (NMF) is an example of a method related to SVD. They are both matrix factorization techniques, however, the conceptual difference is that the low rank matrices U_k and V_k, as explained in Section 2.4.2, are constructed in a way such that there are only positive entries, which explains the name of the method. Also, NMF does not find a Σ matrix that contains singular values on the diagonal. This method gave similar results or worse compared to SVD, which is why it will not be further investigated.

Another example that was investigated is the SVD++ algorithm. This algorithm is similar to SVD, which only takes explicit information into account, while SVD++ also takes the implicit information into account. The reason for doing this is that it is more likely that a user likes an item that they actually rated compared to a random item. More details about SVD++ can be read in [33]. As with NMF, SVD++ did not yield better results compared to regular SVD, and was therefore not further investigated.

2.5 Summary of methods

In this section, some benefits and drawbacks regarding CBF and CF will be presented.

2.5.1 CBF- advantages and disadvantages

In a scenario where the users outnumber the items by far, item-item CBF can easily scale to numerous users. This is due to the fact that the recommendations do not depend on other users, only the item matters. However, if there are a lot of items, finding similar items could take a lot of time.

A major drawback with CBF lies in the content, which often is hand-engineered. For instance, if an item has a vague description, it is unlikely that similar items will get a high similarity score and eventually get recommended. Thus, it is important that the category, tags, description, etc., are well-thought-out. Another problem with CBF lies in the absence of serendipity, which in this context means that the model cannot discover new user interests. This is the result of only using the interaction history of one user at a time, and not others, like CF does.

2.5.2 CF- advantages and disadvantages

CF has many advantages and has in fact been the most commonly used technique when it comes to the implementation of RS [2]. A major benefit with CF is the ability to include serendipity in the recommendations. From a human perspective, seemingly irrational recommendations could be provided by CF. However, as long as similar users have purchased other items, these items have a good chance of being recommended. CBF does not include much serendipity, at least not when implemented in an item-item fashion.

A common problem with CF is referred to as the cold-start problem [52]. New users, i.e., users that have not made their first purchase yet, are the definition of cold starters. Since no information regarding interactions is present, MF models cannot find the user factors. However, there are ways to somewhat get around this problem, as will be discussed in Chapter 4.

2.6 Metrics

There are multiple ways to evaluate a recommender system, and it is not always clear what the best metric would be in a given situation. Schröder et al. [53] introduce four major classes that most metrics belong to: predictive accuracy metrics, classification accuracy metrics, rank accuracy metrics, and non-accuracy metrics. Depending on whether the user feedback is implicit or explicit, different measures are more or less suitable. Before presenting some examples of each category that are used as metrics in this report, some basic quantities are defined below.

True Positives (TP): Correctly classified positive samples
False Positives (FP): Incorrectly classified positive samples
True Negatives (TN): Correctly classified negative samples
False Negatives (FN): Incorrectly classified negative samples

To give an example, looking at the confusion matrix in Table 2.1, a TP would be, in the case of implicit feedback, that we predict an interaction between user u and item i while this is true.

Table 2.1: Confusion matrix, general scheme.

	Predicted:		
True:	interaction	no interaction	Total
interaction	TP	FP	TP + FP
no interaction	FN	TN	FN + TN
Total	TP + FN	FP + TN	n

2.6.1 Root mean squared error

Root Mean Squared Error (RMSE) is a predictive accuracy metric, and it is one of the most commonly used metrics to evaluate a recommender system [7], at least when explicit feedback is available. When only implicit feedback is available, RMSE does not have the same relevance, since the prediction would turn into a binary classification problem instead. For instance, if a prediction for user u on item i is 0.9, and u actually has interacted with i (a score of 1), it would be unfair to interpret this as an error of 0.1.

As the name indicates, RMSE is calculated by taking the sum of the squared errors, i.e., the difference between the observed ratings r_i and the corresponding predicted ratings $\hat{r}_i$, then averaged by dividing with the number of observed ratings N, and finally taking the square root of the average, as

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{N}(r_i - \hat{r}_i)^2}{N}}. \tag{2.13}$$

As can be seen in Eq. 2.13, the errors are squared before they are averaged. This means that a higher weight is given to large errors, i.e., the measure is sensitive to outliers. This is important when large errors are undesirable. RMSE as a measure is intuitive and gives an indication of the quality of the predictions. The lower the value of RMSE, the closer the predicted values are to the observed values. This metric will be used to evaluate SVD in Chapter 4.

2.6.2 Precision@k

Precision@k is an example of a classification accuracy metric, also commonly used to evaluate RS [43]. Ignoring the k at the moment, precision is defined as

$$\text{Precision} = \frac{\sum TP}{\sum TP + FP}. \tag{2.14}$$

Another way of defining precision would be as the proportion of the predicted successes that are true successes, i.e., the proportion of recommended items that are relevant for a user. Following this, precision@k is the proportion of recommended relevant items in the k most confident recommendations. This means that the measure completely ignores what happens *after* k. For instance, if user u gets 5 recommendations where 3 of them are relevant, precision@k would be 0.6, regardless of the relevance of the remaining recommendations, whereas precision would take this into account.

The reason for using precision@k instead of precision is simple. For instance, in a real life situation where a user interacts with some kind of application with a recommender system, lets say YouTube, the user would hardly care about what is *not* shown to them. Even though the engine can produce countless recommendations, with the most confident ones at the top, most recommendations will probably not be displayed anyway. If the 5 most confident recommendations will be displayed to a user, then only these should be taken into account when evaluating the metric while the remaining recommendations, which are not displayed to the user, should not. However, the order of the recommendations is not taken into account with this metric [30], meaning that equal weight is given to each item within the top-k items when evaluating precision@k.

2.6.3 Recall@k

Recall@k is also an example of a commonly used classification accuracy metric [43]. Temporarily ignoring the k, it is defined according to

$$\text{Recall} = \frac{\sum TP}{\sum TP + FN}. \tag{2.15}$$

Introducing the k again, the definition is equivalently described as the proportion of the true successes that have been correctly classified as successes in the top-k candidates, or, equivalently, as the proportion of relevant items that are recommended to a user in the top-k candidates. This metric is useful in combination with precision@k, since precision is somewhat dependent on recall. For instance, if recall gets a score of zero, this would lead to the same score for precision. If no relevant items are in the top-k candidates, then precision@k cannot be anything apart from zero.

2.6.4 Subjective evaluation

This is an example of a non-accuracy metric, since nothing except subjective opinion is used for evaluation. Clearly, this alone cannot be used to motivate why a model is preferred over another, but it can still be useful to look over some recommendations and see if the results look somewhat reasonable. For instance, suppose an algorithm is evaluated as better compared to some baseline, according to precision@k. If the algorithm, due to some reason, predicts the same items to everybody, lets say the most popular items, then this would not be discovered if no subjective assessment was involved. Therefore, as a rule of thumb, one should always look over the recommendations manually.

2.6.5 Other metrics

Any closely related metric of RMSE, e.g., Mean Average Error (MAE), could also be useful for evaluation. However, the objective function in SVD is minimizing the RMSE, which is why this is preferred.

Mean reciprocal rank was another classification metric that was investigated. The metric returns a score of how high the correct interaction is in the rankings of the predictions. As this metric only regards the first relevant prediction, it proved to be a worse alternative to precision@k and was therefore not included in further investigations.

Area under the receiver operating characteristics (ROC) curve (AUC) is the probability that, given a random pair of observations, where one is a true positive (TP) and one is a true negative (TN), the TP has a higher predicted probability of being a true prediction than the TN has. Thus, the AUC gives the probability that the model correctly ranks such pairs of observations, which can be useful to know when evaluating RS. However, AUC was not included in the results as it did not provide any useful information.

2.7 Clustering evaluation methods

There are a few techniques that can be used when evaluating clusters. The ones used in this project are Adjusted Rand index (ARI) and accuracy. ARI compares how similar clusterings are, which is to say that the method is unconcerned with class labelling [17]. When comparing two clusterings, X and Y, with the same number of clusters, of a set S using Rand index, every pair of elements in S is defined as [27]:

- type (i): the elements of the pair are found in the same cluster for X and for Y
- type (ii): the elements of the pair are found in different clusters for X and Y
- type (iii): the elements of the pair are found in different clusters for X and the same cluster for Y
- type (iv): the elements of the pair are found in the same cluster for X and different clusters for Y

The Rand index is then calculated as

$$R = \frac{(i) + (ii)}{(i) + (ii) + (iii) + (iv)} = \frac{number\ of\ agreeing\ pairs}{number\ of\ pairs}. \tag{2.16}$$

The ARI takes into account that both clusterings may not use the same label for the same clusters and uses a random model in order to correct this. A score of 0 corresponds to independent clusterings and 1 corresponds to identical clusterings.

The other metric used was accuracy score for classification, which computes the subset accuracy, i.e., how many of the elements that were correctly labelled for each subset [45] according to

$$accuracy(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n-1} 1(\hat{y}_i = y_i), \tag{2.17}$$

where n is the number of elements, y the true label and $\hat{y}$ is the predicted label.

Chapter 3
Approach

This section contains considerations regarding the data set and the implementations of the models, as well as methods used in succeeding chapters.

3.1 Data description

As stated in Chapter 1, a data set containing various information was provided by WEIQ, given as a dump of relational databases, stored in a Structured Query Language (SQL) file. To be able to read this file in a programming language like Python, we converted it to a Hierarchical Data Format (HDF5) file, and finally, to Pandas Data Frames [37, 56], one for each table. There is no need for describing the whole data set since most of the information provided is probably redundant for a RS. The data frames of interest contain information about users, sellers (the restaurants), items (the products of the restaurants), item categories, and item tags. Table 3.1 shows an example of how a row from the data frame could look like.

Table 3.1: Example of how the data is stored.

user-ref	seller-ref	item-ref	category-ref	tag
b5202025...	3bf6a892...	c09b0c1e...	0ac5832a...	Vegetarian

Each row has a user reference, an item reference, a seller reference, a category reference, and in some cases one or more tags. If one would like to know what the real item, seller or category is, the reference maps to this information in another data frame. The tags are not masked with a reference. However, the users are anonymous as far as the reference goes. This kind of data can be useful as an input to a RS, to analyze the preferences of the users. Table 3.2 illustrates how the data frame in Table 3.1 can be interpreted.

Table 3.2: Interpretation of the data.

user	restaurant	item	category	tag
Anna	Joe's Pizza Place	Margherita	Pizza	Vegetarian

In this example, Anna went to Joe's Pizza Place and bought a Margherita, which is in the category "Pizza" and has the tag "Vegetarian". However, items sometimes comes with a description, which, due to illustration purposes, is not shown in Tables 3.1 and 3.2. As is show in Chapter 4, the descriptions, among with categories and tags, will be used as an input to the content-based RS.

3.2 Overview and statistics

In Table 3.3, some statistics of the data are presented. The first column is regarding the full data set, with over 100 unique sellers and almost 70000 users. In Figure 3.1, the distribution of number of purchased unique items per user is shown.

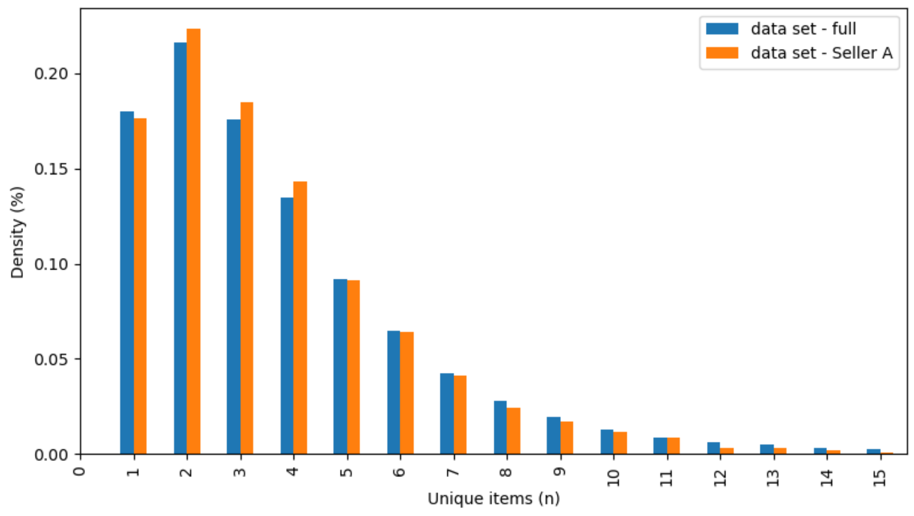


Figure 3.1: Distribution of number of purchased unique items per user for the full data set and Seller A. Note that the figure is cropped due to the small proportion of users who bought more than 15 unique items.

As can be seen in the figure, two distributions are shown at the same time. The blue bars represent the full data set, while the orange represents a specific seller. From now on, the data set of this seller will be referred to as Seller A. We decided to reduce the data set to this particular seller due to several reasons. Firstly, as can be confirmed in Figure 3.2, Seller A has the highest amount of sold items of all restaurants in the full data set. Secondly, Seller A has a wide range of items, see Table 3.3, offering 119 different products. The range of the items somewhat represents the full data set, without risking losing too many important patterns.

Lastly, the distributions of purchased unique items are quite similar for the full data set and Seller A, confirmed in Figure 3.1, but the interaction density of Seller A, as can be seen in the second column of Table 3.3, is almost 50 times higher compared to the full data set. This is the most important reason for choosing Seller A, and we believe this data will be less noisy compared to the full data set, and thereby easier to analyze. If this is not the case, this will confirm that it is unlikely that a model could be estimated for every single restaurant, since this restaurant has one of the best qualifications among all sellers.

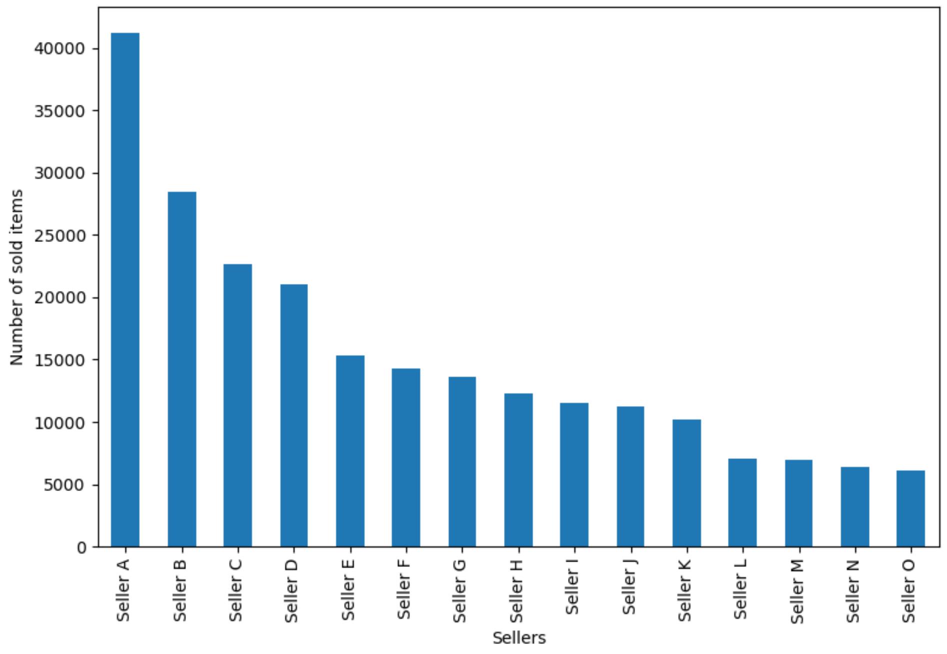


Figure 3.2: The number of sold items for the 15 biggest sellers in the full data set.

Interaction density is calculated according to

$$\rho = \frac{number\ of\ interactions}{number\ of\ users \cdot number\ of\ items} \tag{3.1}$$

which is the same as degree of filling in the interaction matrix. In the field of RS, it is known that it could be hard to extract user behavior if the users have interacted with too few items. Herlocker et al. discuss this and suggest that for MF algorithms an interaction density below 5% could be hard to work with [23]. Unfortunately, the interaction density of both the full data set and Seller A is below 5%.

Furthermore, looking at Figure 3.1, about 40% of the users have only made 1 or 2 purchases, both for the full data set and for Seller A. This could be a major problem, since these users will be cold starters. To be precise, users that have not made their first purchase yet, as explained in Section 2.5.2, which are not included in our data set, are the definition of cold starters. However, users with a few purchases will likely suffer from the same effect and not contribute with valuable data to the RS, at least when working with CF. Therefore, we remove all the users with less than 3 purchases, to somewhat prevent the effect of cold starters. Removing 40% of the data is definitely not desirable, since we from start have very limited data to work with, but to our best understanding, it will do more good than harm.

Table 3.3: Statistics of the full data set and Seller A.

	full data set	Seller A
unique users	68604	6335
unique items	5272	119
unique sellers	113	1
unique categories	239	7
unique tags	348	17
items with tags	67%	81%
items with description	68%	76%
mean amount of items/user	3.8	3.6
σ of items/user	3.0	2.5
max amount of items/user	67	24
interaction density	0.07%	3.8%

3.3 Input data for modelling

When estimating the performance of a model, it is important to make a fair assessment. To do this, the data from which the model-performance is estimated needs to be separated from the data that the model is trained on. Normally, the dataset is divided into a train set and a test set, by some fraction. However, when working with RS and interaction matrices, the way to estimate performance is to mask user-item interactions in the train set, according to Figure 3.3. When the model predicts the user-item interactions, these masked entries should hopefully be included in the top recommended items for the users. One important aspect of this data division is that entries cannot be masked entirely at random, as that could result in some users having one or no interactions with items whatsoever. Therefore, a threshold needed to be utilized, limiting the masking of entries to users that had more interactions than the threshold.

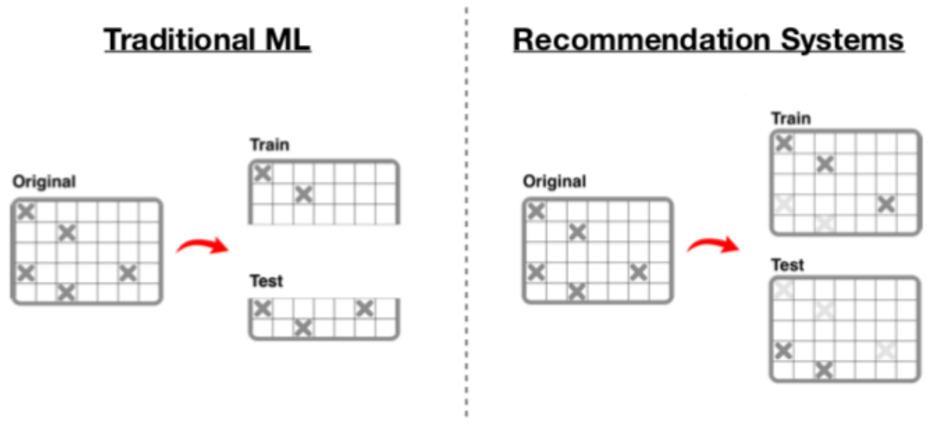


Figure 3.3: Representation of how a data set for RS is split into train set and test set by masking, in comparison to a normal data split in ML. Image from [42].

As will be described in Section 3.8, the models' hyperparameters are optimized. Thus, to keep the integrity of the data, the data is divided into train, validation, and a test set, with the hyperparameters optimized on the validation set. With that said, we decided to mask 30% of the entries, split equally between the validation and test set.

3.4 Item features

Item features are defined by the metadata that describes the items. The metadata for items available in this project, as mentioned in Section 3.1, are categories, tags, and item descriptions. To give an example of how the categories and tags can look like in the WEIQ app, in Figure 3.4, the categories are *ALLA (ALL)*, *MAT (FOOD)* and *DRYCK (DRINKS)*, and the first three tags are *Varmrätt (Main)*, *Förrätt (Starters)* and *Öl (Beer)*. A description can also be seen in the figure for the dish *Örtdresserad lammfile med grönsaker (Filet of lamb with vegetables)*, although, in this case, the description is the same as the name of the dish.

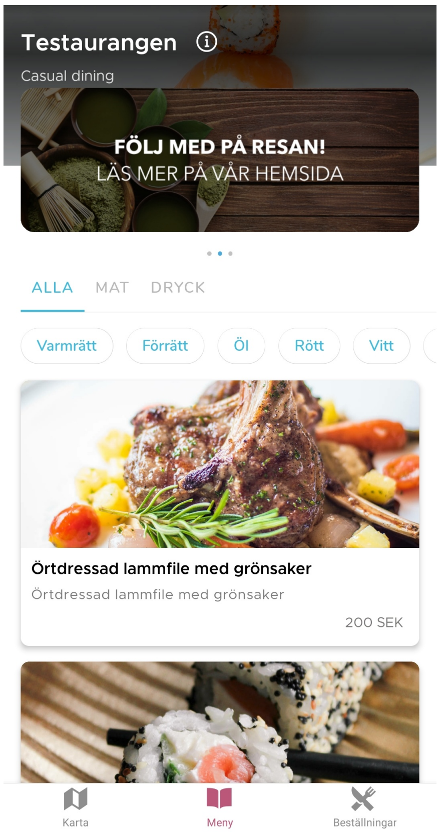


Figure 3.4: An example of how the categories and tags can look like in the WEIQ app.

Categories are supposed to be general and let a user filter between products, for instance, food and drinks. Tags should be more specific and let the user narrow the search after they have chosen a category. For example, a cheeseburger could belong to the category *Food* and have the tag *Hamburger*. As this would give the cheeseburger the same tag as any other type of burger in that restaurant, this information is useful to connect items. Since different restaurants usually have at least some similar items, it would be desirable if they used the same categories and tags for similar items, as this would connect these items between them. However, due to the fact that restaurants write their own categories and tags, this is seldom the case. Furthermore, it could be problematic that every item has a category attached to it, but not always a tag. Since the sellers themselves choose the specific tags and categories, this could lead to redundant categorizations, i.e., that different tags or categories are used by different restaurants to describe the same type of food. In order to appropriately use these categorizations as item features in the RS, they need to be cleaned by removing redundant features that would not give any beneficial information, i.e., to cluster them together. This clustering of item features would not be needed if tags and categories were predefined, e.g., that the sellers would have a finite number of tags and categories to choose from.

3.5 Pseudo explicit ratings

The data we have in this project is implicit feedback. No user has rated any items, only the frequency of user-item interactions is registered. However, we will try two different approaches, one where we treat the data as it is, i.e., implicit, and one where we interpret the data as explicit. For the second approach, we make the assumption that users who bought items several times probably like the items, even though we cannot be certain. Therefore, we convert the interaction matrix to a matrix with what we call a pseudo explicit rating scale from 1-3, where 1 and 2 correspond to 1 and 2 interactions, respectively, and 3 or more interactions are represented by a score of 3. Most users try new items once in a while, which could be unsatisfactory. If this is the case, they would in all likelihood not buy the item again. However, if a user buys an item twice, it is more probable that the user actually likes the item. Finally, buying an item 3 times or more should probably be enough for making the assumption that the user actually prefers the item. With these two types of interaction matrices, we will perform different MF methods, suitable for implicit and explicit feedback, respectively. In addition to this, we will use WARP, as described in Section 2.4.4, on the implicit data set.

3.6 Level of confidence

So far, implicit feedback has been presented as binary, i.e., either an interaction between a user u and an item i is present or not. This is true, however, one could argue that if u interacts several times with i, u probably likes i, even though u has not expressed this explicitly. In fact, the interaction matrix in Figure 2.1 could be seen as implicit, without replacing all numbers that are non-zero with 1. Accordingly, the numerical value of implicit feedback, which represents frequency, could be interpreted as level of confidence [26]. For instance, if u interacts with something only one time, the confidence would be 1. The reason for this could be that u bought the item for a friend, u tried a new item they did not like, or simply that u bought the item, was happy with it and still never bought it again. The point is, as stated before, that implicit feedback does not tell the user preference, however, it is more likely that frequently occurring interactions with the same items reflect the user opinion.

3.7 Inverse propensity weights

Popular items are the ones most frequently interacted with by the users. The problem with popular items is that they tend to outscore most of the other items when making recommendations. The result is that everyone gets recommended the same popular product. One solution to this is to weigh down the popular items in the interaction matrix, in order to get less biased recommendations. The weighing method used is Inverse Propensity Weighing [9]. Having an interaction matrix, the weighing function takes the total number of users (rows) in $\boldsymbol{U}$ divided by the number of the users that have interacted with item i [35]. To give an example, having an interaction matrix $\boldsymbol{A}$ and applying the weighing function, followed by a normalization between 0 and 1, would result in

$$A = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix} \xRightarrow{\mathcal{W}(A)} \begin{pmatrix} 0 & 0 & 4/1 \\ 4/3 & 4/2 & 0 \\ 4/3 & 0 & 0 \\ 4/3 & 4/2 & 0 \end{pmatrix} \xRightarrow{Normalize} \begin{pmatrix} 0 & 0 & 1 \\ 1/3 & 1/2 & 0 \\ 1/3 & 0 & 0 \\ 1/3 & 1/2 & 0 \end{pmatrix}. \tag{3.2}$$

The augmented interaction matrix obtained is now unbiased toward item frequency. Results concerning this method can be found in Section 4.

3.8 Hyperparameter optimization

Hyper-parameters are parameters whose function is to decide the model selection by controlling the learning process of the model [6]. They determine for instance the regularization and learning rate of an algorithm and are later fixed during the fitting of the model.

Hyper-parameter optimization is the process of identifying values for the hyper-parameters of the model that will minimize/maximize the objective function [5]. Our objective is to find the hyper-parameter setup that maximizes precision@k. The optimization function [12] is defined as

$$\theta^* = \underset{\theta}{\text{argmax}} C(p_{te}, q_{te}, r_{te}, M(p_{tr}, q_{tr}, r_{tr}, \theta)) \tag{3.3}$$

where $\boldsymbol{C}$ is the function determining precision@k, $\boldsymbol{M}$ is the modeling function. $\boldsymbol{\theta}$ contains the hyper-parameters. $\boldsymbol{p_{tr}}$ and $\boldsymbol{q_{tr}}$ are the item- and user-features vectors for the training set, and $\boldsymbol{p_{te}}$ and $\boldsymbol{q_{te}}$ are the item- and user-features vectors for the test set. $\boldsymbol{r_{tr}}$ and $\boldsymbol{r_{te}}$ are the true interaction vectors.

There are several ways to search for the optimal hyper-parameter setup, but two common ones are grid search and random search. When using grid search, a range of values is manually chosen for each hyper-parameter. This set of ranges make up a grid of hyper-parameter values. Each sampled parameter setup of the grid is iterated, and a precision@k score is obtained. The setup which gained the highest score is then assumed to be the optimal hyper-parameter setup. Random search functions similarly to grid search, however instead of manually choosing the hyper-parameter values, a distribution is chosen for each hyper-parameter. As shown in appendix A.1, the hyper-parameters available for optimization differ for the models based on the framework used to implement the model. The effectiveness of hyper-parameter optimization on our model is further discussed in Chapter 4.

Chapter 4

Results and discussion

4.1 Item feature clustering using NLP

Due to the length and variation of the quality of the item descriptions, they were only included as item features for the CBF. As can be seen in Table 3.3, there are 239 unique categories and 348 unique tags in the full data set. Two approaches to cluster the tags and categories were proposed. The first was an ad-hoc approach, to manually classify the tags and categories, in order to reduce the redundances. This approach, while also being time-consuming, would not be feasible outside this project, since the sellers would still be able to create new categories and tags, which in turn would dilute the set and reduce the quality of the item features. The second approach is to use the unsupervised clustering method, k-means, as described in Section 2.2.2. Some preprocessing was done on the categories and tags to reduce the redundant tags, for example by removing unnecessary spaces, symbols, and lowering capital letters. After the preprocessing step, 31 excessive categories and 41 excessive tags were removed. The manually classified tags were divided into three classes, respectively: *food*, *drink*, and *miscellaneous* (*misc*), summarized in Table 4.1. The misc class contains the categories and tags that are unrelated to food or beverages, e.g., *Children*, which from the context could mean children's menu.

Table 4.1: The three classes the categories and tags were manually divided into, used for classification.

type	food	drink	misc	total
Categories	88	59	61	208
Tags	141	123	44	307

4.1.1 Clustering of tags

The tags were clustered using k-means, as described in Section 2.2.2. Three clusters are compared to the labelled classes presented in Table 4.1.

The results from the tag clustering are presented as a confusion matrix in Table 4.2, and as scatter plots in Figure 4.1. As can be deduced from the confusion matrix, the majority of the *food* and *misc* tags are appropriately classified, while *drink* is mostly misclassified as *misc*. Similarly, the misclassified *food* tags were also classified as *misc*. This is reasonable given the fact that there are three clusters and that the *misc* class contains tags that are vague, making its cluster vaster than the others. The scatter plot to the left and right in Figure 4.1 displays the manually assigned tags and the k-means clustering of the tags, respectively. The use of more than 3 clusters was investigated, however, due to several outliers, the k-means algorithm made each outlier into a singular cluster.

The overall accuracy of the clustering was 0.62 and the adjusted rand score was 0.29, both scores being low. The overlapping clusters are not necessarily an issue, as items should be able to have several tags. However, due to the unsatisfactory scores, we decided against implementing these clustered tags in the item features used by the models.

Table 4.2: Confusion matrix of the labelled tags and the predicted labels clustered using k-means.

		Predicted		
		food	drink	misc
Actual	food	100	0	41
	drink	4	57	62
	misc	7	3	34

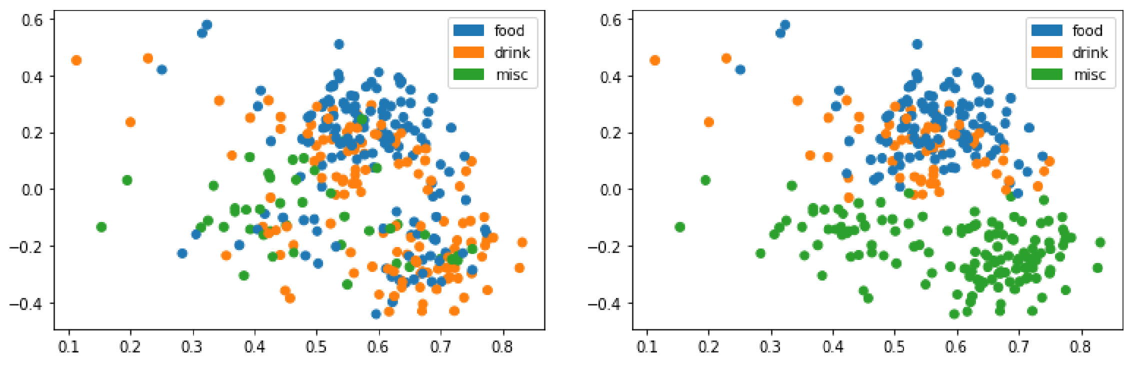


Figure 4.1: Scatter plots of the tags in the 2-dimensional vector space. The left plot displays the manually assigned tags for the original tags and the right-most plot shows the k-means clustering of the tags.

4.1.2 Clustering of categories

The clustering of the categories was conducted in the same fashion as the clustering of the tags, using k-means and the manually labelled classes displayed in Table 4.1. As can be seen in the table, a larger portion of the categories are labelled misc than for tags, which presumably will make this clustering more difficult.

The clustering of the categories is presented in the confusion matrix in Table 4.3 and the scatter plots in Figure 4.2. Judging from the confusion matrix, there are more misclassifications overall. Approximately half of the predicted *food* and *drink* categories are misclassified as *misc*, and 35 of 61 *misc* labels are misclassified as *food*. The same issues that were discussed previously about the clustering of tags are present here as well but to a greater extent. In the left scatter plot of Figure 4.2 there is no clear structure to the classes, i.e., a great amount of overlapping, which in turn makes it difficult to cluster the categories satisfactorily. The resulting accuracy score was 0.47 and the adjusted rand score was 0.14, both of which also condemns the k-means clustering of categories. For the same reason as with the clustering of tags, but to a greater degree, we concluded that the results were too unsatisfactory to consider using these clusters as item features.

Table 4.3: Confusion matrix of the labelled categories and the predicted labels clustered using k-means.

		Predicted		
		food	drink	misc
Actual	food	42	6	40
	drink	0	31	27
	misc	35	1	25

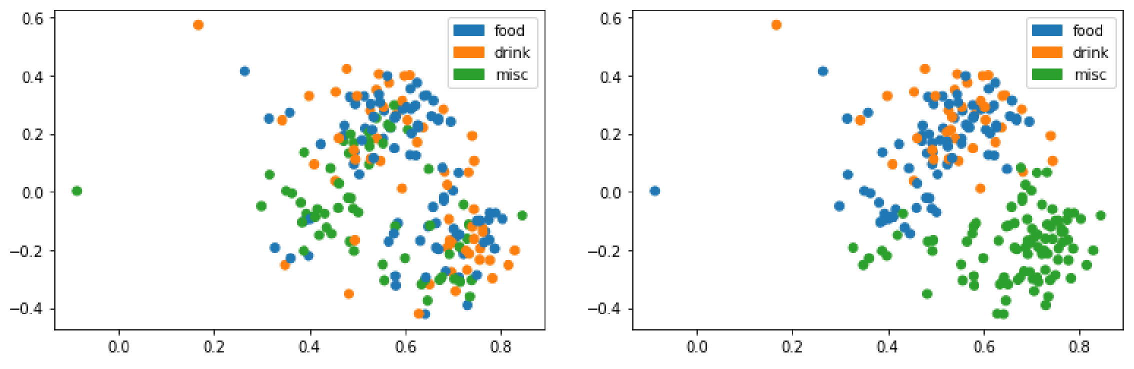


Figure 4.2: Scatter plots of the categories of the full set in the 2-dimensional vector space. The plots are showing the dimensionality of the embeddings reduced with singular value decomposition. The clustering method is k-mean with three clusters.

4.2 Content-based filtering

Due to the lack of demographic information about the users, as mentioned in Section 2.3, this method was implemented in an item-item fashion, independent of which user the recommendation is based on. Therefore, there is no good way to make an objective assessment of this method, since all the users will get the same recommendations for the same item. In Table 4.4, the top 5 recommendations together with their similarity score are shown for a couple of frequently sold items from Seller A. The similarity scores were calculated from a TFIDF representation of the words in the documents. In this case, the documents consisted of the tags, categories, and item descriptions, in a bag-of-words representation. To be clear, the same bag was used, not several with different weights. Another option would be to use three different bags and put different weights on them. However, this was never further investigated, due to the varying quality or absence of tags and item descriptions.

Table 4.4: Recommendations made for 3 frequently bought items from Seller A, together with its similarity score, using CBF.

Husets burgare	Miss Behave IPA	Coca Cola
Thempehburgare, 0.32	Cirrus Cloudy Lager, 1.0	Coca cola Zero, 1.0
Tropical burger, 0.31	Kona Fire Rock, Pale Ale, 0.80	Fanta, 1.0
Sweet potato fries, 0.13	Kona Big wave, 0.80	Loka, 1.0
Räkfrossa, 0.11	Kona Hana Lei IPA, 0.80	Sprite, 1.0
Popcorn, 0.10	Kona Wheat Ale, 0.80	Codorniu alkfritt, 0.63

As explained above, subjective assessment has to be used in this case, and it is hard to say if the recommendations are good or not, although most of them have something in common with their respective item. However, remarks can be made regarding the scores. For example, looking at the recommendations for *Husets burgare*, the scores are significantly lower compared to the recommendations made to the beer and soda. A longer description is present for all the recommendations made from Husets burgare, which explain why the scores are lower, compared to a short or no description for the other recommendations. In the case when the recommended items get a score of 1.0, the same, or, more likely, no description at all is present, together with the same category and tags. Considering how the cosine similarity of the document vector is calculated, described in Section 2.2.4, the same description, category, and tag will of course give the highest score of 1.0. However, when there exist a description containing a couple of words or more, it gets more difficult to match all the words and get a higher score. This does not mean that a thorough item description, which probably is more than a few words long, is a bad thing. The better the description, the more we can distinguish the items.

4.3 Pseudo explicit filtering

As is common when working with ML, a naive predictor, called a baseline, is often used to have a reference to compare the predictions of the models to. Our first thought was to compare with a random predictor, which randomly predicts the rating r_{ui} user u gave to item i, on the test set between 1 and 3 without using any information or context. We realized that it is probably easy to beat such a predictor, and that better predictors for comparison can be constructed. If we instead compare with the global mean rating μ, a better baseline is achieved. This would be a fairly reasonable predictor, since it takes all the rating in the data set into account. Some users tend to rate items differently than other users, despite the fact that both liked or disliked the item. Therefore, it is reasonable to also include the user bias b_u in the predictor. The same reasoning goes for the items, since some items tend to have higher ratings than other items, due to the fact that some items are more popular than other items, which is why the item bias b_i also should be included. As a final attempt to improve our comparative predictor, we construct the baseline rating b_{ui} as

$$\hat{r}_{ui} = b_{ui} = \frac{\mu + b_u + b_i}{3} \tag{4.1}$$

which takes the global mean, the user bias, and the item bias into account.

In Figure 4.3, the RMSE using SVD and the baseline, for the full data set and Seller A is displayed, as a function of latent factors. Note that the baseline is constant, and does not depend on the number of factors.

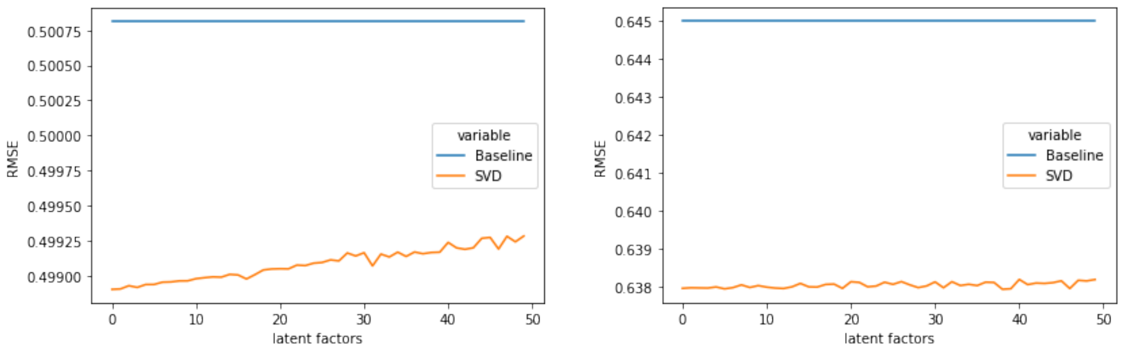


Figure 4.3: RMSE from pseudo explicit feedback, for the full data set (left), and Seller A (right).

The reason for displaying the RMSE for different latent factors is that, as explained in Section 2.4.1, we seek the least number of factors to explain as most variability as possible. Normally, we would expect the curve to look more like a positive quadratic function, i.e., that the RMSE would start at some value, decrease until the best number of latent factors is found, and then increase again. As can be seen in the figure, the curve looks more like a slightly sloping straight line, both for the full data set (left) and Seller A (right). The RMSE does not decrease for any factor but 1. In practice, 1 latent factor is essentially representing the most popular item [29], which certainly makes an algorithm like SVD unnecessary to use in this case.

In Section 3.5, we introduced the concept of pseudo explicit interpretation of the data. From the beginning of this project, we assumed that this approach would be reasonable. However, the simple one-to-one mapping between interaction and preference that we assumed may be too simplistic [41, 44]. The previous work regarding converting implicit feedback to explicit suggests some form of parametric mapping. This is something that would have been more thoroughly investigated if we knew this from the start and had more time. It is hard to say if the potentially simplistic mapping is the reason for the SVD not performing significantly better compared to the baseline, or that the interaction density is too low to find any patterns. Furthermore, using the rating prediction metric RMSE to achieve satisfactory recommendations may not be the ideal way to go. The task for rating prediction metrics is to inspect the rating precision on the entire data set, which does not favor the top k recommendations and therefore will not optimize the model for the recommendations relevant for our task. Nonetheless, a pseudo explicit interpretation of the implicit data set is, judging from the poor results and the reasons stated, not a good enough approach for this problem.

4.4 Implicit filtering

The baseline for implicit filtering was not constructed in the same way as for explicit filtering, with an average of the global mean μ, the user bias b_u, and the item bias b_i. This is due to the fact that implicit feedback does not incorporate rating in the same way as explicit feedback does. Instead, as a baseline, we used the WARP and ALS algorithm with only one latent factor, respectively, essentially corresponding to the most popular item, as described in Section 4.3. Since the baseline only uses one latent factor, it is constant.

4.4.1 Implicit filtering with WARP

Figure 4.4 displays precision@k (left) and recall@k (right) against latent factors for the WARP-model with item features, the WARP-model with inverse propensity weights, and the baseline for the full data set. This time, the results look more like expected, i.e., as a negative quadratic function, compared to the results for SVD in Section 4.3 (in contrast to RMSE, precision@k and recall@k is suppose to be as high as possible, which is why a negative quadratic function instead of a positive is expected). As can be seen in the figure, the WARP-model with inverse propensity weights performs the best. Precision@k and recall@k are strongly correlated. Both of the models outperform the baseline and the optimal number of latent factors for the WARP-model and the inverse propensity weighted WARP-model are approximately 10 and 20, respectively. The hyper-parameters for the models can be found in Table A.1 in Appendix A.1.

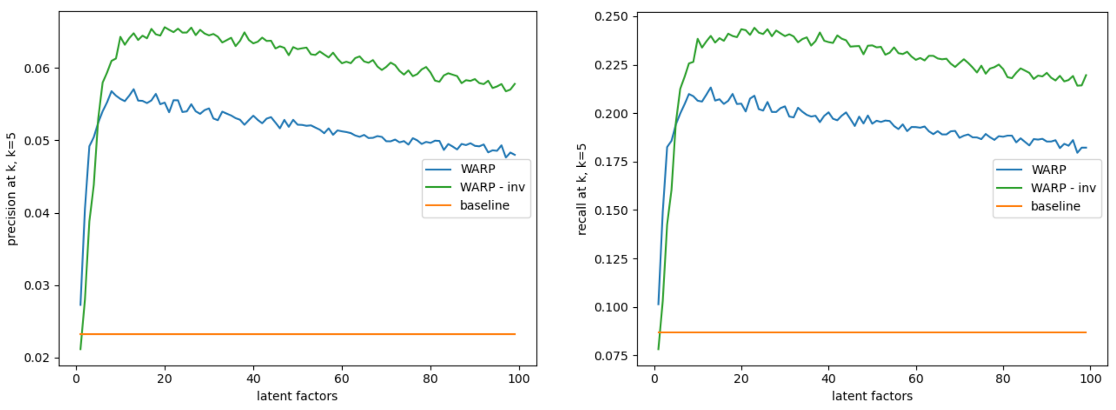


Figure 4.4: Precision@k (left) and recall@k (right) from implicit feedback for the full data set. The blue curve represents the WARP-model, and the green curve represents the WARP-model with the inverse propensity weights-implementation, and the orange curve represents the baseline.

Figure 4.5 displays the same scores and models as above for Seller A. The baseline proved hard to outperform for the WARP-models. In contrast, the baseline predictor for the full data set was significantly easier to outperform. The reason behind this, we assume, is that the popular items of the whole data set are limited to the items from several popular restaurants. Accordingly, the baseline predictor for Seller A is much better, as it recommends the popular items from only one restaurant.

Both models decline rapidly, especially the standard WARP-model which immediately drops below the baseline. The linear decline is understandable as there is no valuable information contained in the less significant latent factors, i.e., the model is overfitting. The initial quadratic slope does, however, indicate that there is some kind of important pattern in the more significant latent factors, but it is strange that these worsen the score (almost like an anti-correlation). This rapid decline may be due to the data set being too small, resulting in there being no significant pattern aside from popularity. Solely based on these metrics, it is difficult to determine how many latent factors that would ultimately give the best recommendations for the users. Intuitively the best number of latent factors to use would be 5 and 15, for the WARP-model and the inverse propensity weighted WARP-model respectively, as at approximately these numbers of latent factors the curves decline linearly. However, based on the precision@k and recall@k the optimal number of latent factors would be 1. It would be appropriate to evaluate the top-k recommendations subjectively instead.

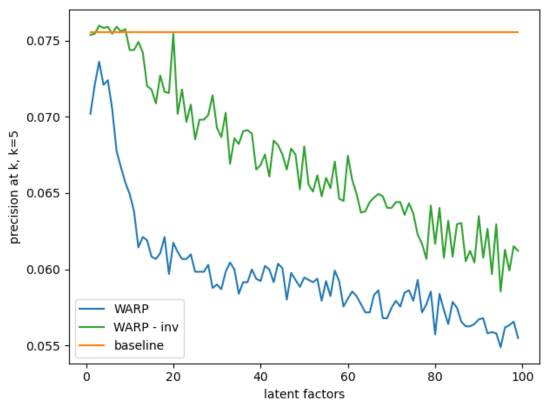


Figure 4.5: Precision@k (left) and recall@k (right) from implicit feedback for Seller A. The blue curve represents the WARP-model, and the green curve represents the WARP-model with the inverse propensity weights-implementation, and the orange curve represents the baseline.

4.4.2 Implicit filtering with ALS

In Figure 4.6, precision@k (left) and recall@k (right) using ALS and a baseline for the full data set is displayed, as a function of latent factors. Similarly to the results concerning WARP in Section 4.4.1, these results also look more like expected. As can be seen in the figure, ALS significantly outperforms the baseline. Again, precision@k and recall@k are strongly correlated, which is why the curves look similar. The optimal number of latent factors seem to be at approximately 35. The hyper-parameters for the model can be found in Table A.1 in Appendix A.1.

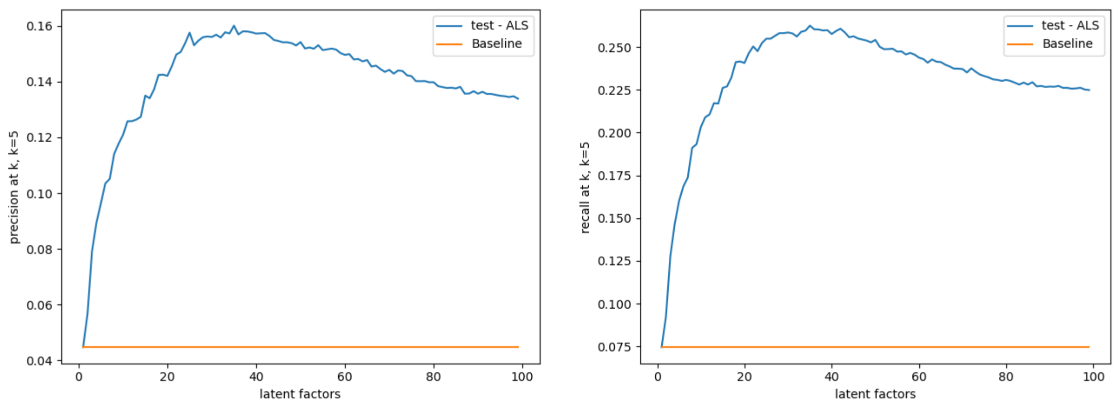


Figure 4.6: Precision@k (left) and recall@k (right) from implicit feedback for the full data set.

In Figure 4.7, precision@k (left) and recall@k (right) using ALS and the baseline for Seller A is displayed, as a function of latent factors. A bit surprising, the results for Seller A are worse compared to the full data set, at least when looking at the shape of the curves.

It does not seem like the ALS can beat the baseline, and both precision@k and recall@k decreases with the number of factors. Essentially this means that the use of ALS on this data set is unnecessary according to these metrics. Important to note here is that the baseline is much higher for Seller A compared to the full data set, which might be due to the same reason mentioned in Section 4.4.1. The hyper-parameters for the model can be found in Table A.2 in Appendix A.1.

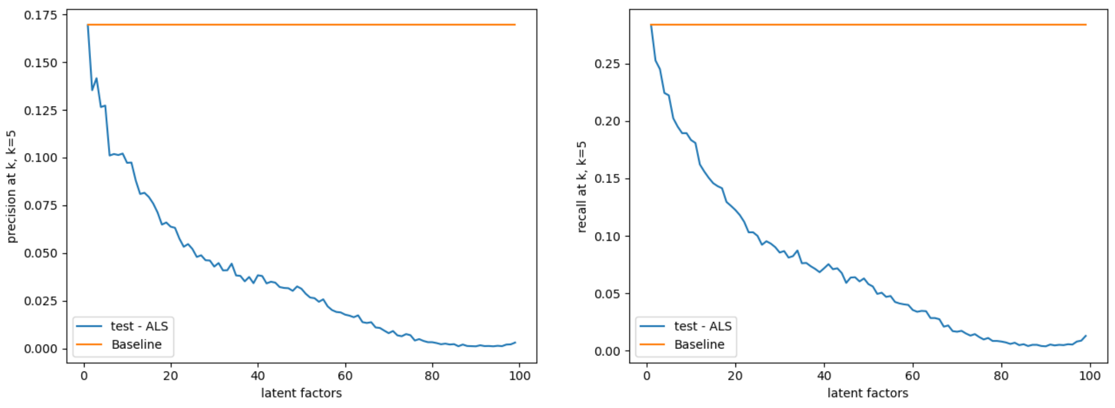


Figure 4.7: Precision@k (left) and recall@k (right) from implicit feedback for Seller A.

4.5 Subjective evaluation

In Appendix B.1 and B.2, tables regarding recommendations for two users u_1 and u_2 are made, regarding the full data set and Seller A, respectively. The different tables contain the true interactions, i.e., what the users have bought previously, and the recommendations made by the different algorithms, including the baseline. In the example of recommendations for the full data set, in Tables B.1 and B.2, we see that the ALS-model and the WARP-model with inverse propensity weights gave similar and relevant recommendations, while the normal WARP-model recommended different items, although still somewhat relevant. For u_2 in Tables B.3-B.4, it is more difficult to determine which recommendations are better suited for the user. The WARP-models gave similar recommendations, while the ALS-model gave entirely different recommendations. This is in line with our previous results, that the models performed poorly for Seller A.

Chapter 5
Conclusion and future work

Overall, the results achieved in this project are a bit ambiguous, judging from the objective results. Despite the fact that the baseline predictions achieve higher scores for Seller A's data set, this does not necessarily mean that the recommendations are satisfactory, instead it may suggest that the metrics when evaluating for such a small set are too biased toward popularity. Even in a situation where promising results are achieved, this does not guarantee that the RS will be useful in reality. The only way to know if the RS, for instance, at an e-commerce web page, actually creates value for the users, is to monitor and collect statistics from the user interactions with the displayed recommendations.

As previously mentioned in Section 4.3 the SVD-model was disregarded as both the results were unsatisfactory, and the RMSE was not an appropriate metric for evaluating top-k recommendations. When comparing the results for the implicit models, both behaved similarly for both data sets, although the ALS-model scored significantly higher than the WARP-model. This is due to the different frameworks being used for the respective models and their evaluation methods, which implemented the precision@k metrics slightly differently. The performance of the models on Seller A's data set was deemed worse, as none of the models outperformed the baseline. While surprising, as we thought the higher interaction density would result in better performance overall, the trade-off of having a smaller data set was so significant that there were no other latent factors than popularity to be considered.

As discussed in Section 2.5, the cold-start problem will occur for CF when new users, which have no interactions with items, are added to the RS. One possible solution to this could be to simply recommend the popular items of the restaurant. Then, after the first purchase was made, regardless of the item bought was from the popular items recommendation or not, CBF could be used for recommending products based on the previously bought item. Again, it does not matter if the user purchases an item belonging to the list of recommendations or not, the important thing is that the user gets some interactions with items. After a couple of interactions from the new user, CF will overcome the cold-start problem and can start to make personalized recommendations, instead of popular- or similar items recommendations. Note that this would not have helped us in this project, although, it is a suggestion to make some form of recommendation to cold-starters in a CF system, without using actual CF.

A collection of data sets called MovieLens, created by GroupLens Research [22], has become popular for research purposes. In these data sets, all the users have rated at least 20 different movies. There is a major difference between the MovieLens data set, and the one used in this project. It would, most likely, have been easier to extract information if there was more data available, and especially, a higher interaction density. The data set provided by WEIQ is also unique in the sense that the users cannot, in practice, access every item at all times, as they are geographically restricted.

In conclusion, the data set is too small to be able to make a satisfactory RS, even with the different improvements and approaches used. There are, however, areas of improvement that are worth considering in the future.

5.1 Future work

Popularity-based cold-start recommendations, as discussed previously in this section, are not personal. A suggestion to make them somewhat personal would be to introduce new users to the app by giving them the option to answer a short survey about their preferences. For instance, letting them choose between meat, chicken, vegetarian, and vegan. If a user chooses chicken, out of the popularity-based recommendations, the chicken related dishes could be displayed. However, important to note is that users may not want to be kept from using the application by answering a survey.

Another thing that could improve the RS with the help of an application feature is to use users search queries and implement them into the RS. The user input into the search field will likely be some item feature, e.g, a tag, and could therefore be used to help improve the recommendations.

As stated in Section 4.2, the CBF was implemented in an item-item fashion, independent of the users. While this is true, there are ways to make it user specific, and thereby, an objective assessment could be done. One could make the CBF user specific by simply calculating the TFIDF representation of a user's previously bought items in the train set, taking the average of this vector and repeat the same procedure as before, i.e., compute the cosine similarity between this vector and every item that is not seen in the train set. Finally, the items with the highest scores could be recommended, and compared to the items the user already interacted with in the test set.

Regarding the way categories and tags are chosen by the sellers, there is room for improvement. If the categories and tags to choose from were predefined, as stated in Section 3.4, this would lead to more consistency. It would be desirable if this is the case in the future, as it probably is easier to relate different items for the CBF and improve item feature usage in the WARP hybrid model.

References

[1] Gediminas Adomavicius and Alexander Tuzhilin. “Context-aware recommender systems”. In: *Recommender systems handbook*. Springer, 2011, pp. 217–253.

[2] Abhinav Ajitsaria. *Build a Recommendation Engine With Collaborative Filtering*. 2019.

[3] Joeran Beel et al. “Research-paper recommender systems : a literature survey”. In: *International Journal on Digital Libraries* 17.4 (2016), pp. 305–338.

[4] James Bennett and Stan Lanning. *The Netflix Prize*. 2007.

[5] James Bergstra and Yoshua Bengio. “Random search for hyper-parameter optimization.” In: *Journal of machine learning research* 13.2 (2012).

[6] James Bergstra et al. “Algorithms for hyper-parameter optimization”. In: *25th annual conference on neural information processing systems (NIPS 2011)*. Vol. 24. Neural Information Processing Systems Foundation. 2011.

[7] Jesus Bobadilla et al. “A framework for collaborative filtering recommender systems”. In: *Expert Systems with Applications* 38.12 (2011), pp. 14609–14623.

[8] Piotr Bojanowski et al. “Enriching Word Vectors with Subword Information”. In: (2016).

[9] Stephen Bonner and Flavian Vasile. “Causal Embeddings for Recommendation”. In: *CoRR* (2017).

[10] John S. Breese, David Heckerman, and Carl Kadie. *Empirical Analysis of Predictive Algorithms for Collaborative Filtering*. 2013.

[11] Robin Burke, Alexander Felfernig, and Mehmet Göker. “Recommender Systems: An Overview”. In: *Ai Magazine* 32 (Sept. 2011), pp. 13–18.

[12] Simon Chan, Philip Treleaven, and Licia Capra. “Continuous hyperparameter optimization for large-scale recommender systems”. In: *2013 IEEE International Conference on Big Data*. IEEE. 2013, pp. 350–358.

[13] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. “Are We Really Making Much Progress? A Worrying Analysis of Recent Neural Recommendation Approaches”. In: *Proceedings of the 13th ACM Conference on Recommender Systems*. RecSys ’19. Copenhagen, Denmark: Association for Computing Machinery, 2019, pp. 101–109.

[14] Google Developers. *Machine Learning Crash Course*. 2018.

[15] Jingtao Ding et al. “Improving Implicit Recommender Systems with View Data.” In: *IJCAI*. 2018, pp. 3343–3349.

[16] Simon Funk. *Netflix Update: Try This at Home*. Dec. 2006.

[17] Alexander J Gates and Yong-Yeol Ahn. *The Impact of Random Models on Clustering Similarity*. 2017.

[18] David Goldberg et al. “Using Collaborative Filtering to Weave an Information Tapestry”. In: *Commun. ACM* 35.12 (Dec. 1992), pp. 61–70.

[19] Carlos A. Gomez-Uribe and Neil Hunt. “The Netflix Recommender System: Algorithms, Business Value, and Innovation”. In: *ACM Trans. Manage. Inf. Syst.* 6.4 (Dec. 2016).

[20] Edouard Grave et al. “Learning Word Vectors for 157 Languages”. In: *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*. 2018.

[21] Jiawei Han, Micheline Kamber, and Jian Pei. “2 - Getting to Know Your Data”. In: *Data Mining (Third Edition)*. Ed. by Jiawei Han, Micheline Kamber, and Jian Pei. Third Edition. The Morgan Kaufmann Series in Data Management Systems. Boston: Morgan Kaufmann, 2012, pp. 39–82.

[22] F. Maxwell Harper and Joseph A. Konstan. “The MovieLens Datasets: History and Context”. In: *ACM Trans. Interact. Intell. Syst.* 5.4 (Dec. 2015).

[23] Jonathan L. Herlocker et al. “Evaluating Collaborative Filtering Recommender Systems”. In: *ACM Trans. Inf. Syst.* 22.1 (Jan. 2004), pp. 5–53.

[24] Geoffrey Hinton and Sam Roweis. “Stochastic Neighbor Embedding”. In: 15 (June 2003).

[25] Thom Hopmans. *A recommendation system for blogs: Setting up the prerequisites (part 1)*. 2015.

[26] Yifan Hu, Yehuda Koren, and Chris Volinsky. “Collaborative Filtering for Implicit Feedback Datasets”. In: *2008 Eighth IEEE International Conference on Data Mining*. 2008, pp. 263–272.

[27] Lawrence Hubert and Phipps Arabie. “Comparing partitions”. In: *Journal of classification* 2.1 (1985), pp. 193–218.

[28] Hosein Jafarkarimi. “A naive recommendation model for large databases”. In: *International Journal of Information and Education Technology* 2 (June 2012).

[29] Dietmar Jannach et al. “What Recommenders Recommend – An Analysis of Accuracy, Popularity, and Sales Diversity Effects”. In: *User Modeling, Adaptation, and Personalization*. Ed. by Sandra Carberry et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 25–37.

[30] Kalervo Järvelin and Jaana Kekäläinen. “IR Evaluation Methods for Retrieving Highly Relevant Documents”. In: *SIGIR Forum* 51.2 (Aug. 2017), pp. 243–250.

[31] Joseph A Konstan and John Riedl. “Recommender systems: from algorithms to user experience”. In: *User modeling and user-adapted interaction* 22.1 (2012), pp. 101–123.

[32] Yehuda Koren. “Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model”. In: New York, NY, USA: Association for Computing Machinery, 2008.

[33] Yehuda Koren. “Factorization meets the neighborhood: a multifaceted collaborative filtering model”. In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2008, pp. 426–434.

[34] Yehuda Koren, Robert Bell, and Chris Volinsky. “Matrix factorization techniques for recommender systems”. In: *Computer* 42.8 (2009), pp. 30–37.

[35] Dawen Liang, Laurent Charlin, and David M Blei. “Causal inference for recommendation”. In: *Causation: Foundation to Application, Workshop at UAI. AUAI*. 2016.

[36] Ian MacKenzie, Chris Meyer, and Steve Noble. *How retailers can keep up with consumers*. 2013.

[37] Wes McKinney. “Data Structures for Statistical Computing in Python”. In: *Proceedings of the 9th Python in Science Conference*. 2010, pp. 56–61.

[38] Tomas Mikolov et al. “Advances in Pre-Training Distributed Word Representations”. In: *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*. 2018.

[39] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013.

[40] Silvia Milano, Mariarosaria Taddeo, and Luciano Floridi. *Recommender Systems and their Ethical Challenges*. Apr. 2019.

[41] David M. Nichols. “Implicit rating and filtering”. In: *In Proceedings of the Fifth DELOS Workshop on Filtering and Collaborative Filtering*. 1998, pp. 31–36.

[42] Parul Pandey. *Recommendation Systems in the Real world*. 2019.

[43] Giorgos Papachristoudis. *Popular evaluation metrics in recommender systems explained*. Apr. 2019.

[44] Denis Parra et al. “Implicit feedback recommendation via implicit-to-explicit ordinal logistic regression mapping”. In: *Proceedings of the CARS-2011* 5 (2011).

[45] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[46] István Pilászy, Dávid Zibriczky, and Domonkos Tikk. “Fast Als-Based Matrix Factorization for Explicit and Implicit Feedback Datasets”. In: *Proceedings of the Fourth ACM Conference on Recommender Systems*. RecSys ’10. Barcelona, Spain: Association for Computing Machinery, 2010, pp. 71–78.

[47] Anand Rajaraman and Jeffrey David Ullman. “Data Mining”. In: *Mining of Massive Datasets*. Cambridge University Press, 2011, pp. 1–17.

[48] “Latent Factor Models and Matrix Factorizations”. In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 571–571.

[49] Badrul Sarwar et al. “Item-based collaborative filtering recommendation algorithms”. In: *Proceedings of the 10th international conference on World Wide Web*. 2001, pp. 285–295.

[50] Ben Schafer et al. “Collaborative Filtering Recommender Systems”. In: Jan. 2007.

[51] J. Ben Schafer, Joseph A. Konstan, and John Riedl. “E-Commerce Recommendation Applications”. In: *Data Mining and Knowledge Discovery* 33.4 (2001), pp. 115–153.

[52] Andrew I. Schein et al. “Methods and Metrics for Cold-Start Recommendations”. In: 2002.

[53] Gunnar Schröder, Maik Thiele, and Wolfgang Lehner. “Setting Goals and Choosing Metrics for Recommender System Evaluations”. In: 811 (Jan. 2011).

[54] Amit Singhal and I. Google. “Modern Information Retrieval: A Brief Overview”. In: *IEEE Data Engineering Bulletin* 24 (Jan. 2001).

[55] J. Solsman. *YouTube’s AI is the puppet master over most of what you watch*. 2018.

[56] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020.

[57] Ulrike Von Luxburg. “A tutorial on spectral clustering”. In: *Statistics and computing* 17.4 (2007), pp. 395–416.

[58] Jun Wang, Arjen P De Vries, and Marcel JT Reinders. “Unifying user-based and item-based collaborative filtering approaches by similarity fusion”. In: *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*. 2006, pp. 501–508.

[59] Jason Weston, Samy Bengio, and Nicolas Usunier. “Wsabie: Scaling up to large vocabulary image annotation”. In: (2011).

[60] Shuai Zhang et al. “Deep learning based recommender system: A survey and new perspectives”. In: *ACM Computing Surveys (CSUR)* 52.1 (2019), pp. 1–38.

Appendices

Appendix A
Hyperparameters

A.1 Optimal hyperparameter setup

Table A.1: The optimal hyperparameter setup (x = available) for the models trained on the full data set.

model	latent factors	epochs	regularization term	learning rate	item α	user α
SVD	5	20	0.4	0.01	-	-
WARP	39	26	-	0.027	4.17e-8	4.35e-8
WARP inv	21	20	-	0.018	1.19e-8	1.25e-8
ALS	39	24	0.011	-	-	-

Table A.2: The optimal hyperparameter setup (x = available) for the models trained on the Seller A data set.

model	latent factors	epochs	regularization term	learning rate	item α	user α
SVD	10	20	0.4	0.05	-	-
WARP	19	25	-	0.015	6.13e-9	1.21e-8
WARP inv	41	14	0.014	-	1.83e-9	3.09e-9
ALS	44	21	0.008	-	-	-

Appendix B
Example of recommendations

B.1 Example of recommendations- full data

Table B.1: Recommendations (rec.) for user u_1 from the full data set, based on all the items bought by user u_1, using WARP with and without inverse propensity weights, respectively.

True interactions	rec. WARP	rec. WARP inv	rec. baseline
Fidel castro	Sol	Mojito	Öl
Filippa fizz	Husets lager	Vodka redbull	Extra lantpommes
P2	Gin & Tonic	Hot shot	Vatten
Melleruds pilsner eko	Briska pear cider	Gin & tonic	Vitt vin
Razz & sprite	Cola zero	Frozen daiquiri	Mjukglass
Tom collins			
Rom & cola			

Table B.2: Recommendations (rec.) for user u_1 from the full data set, based on all the items bought by user u_1, using ALS.

True interactions	rec. ALS	rec. baseline
Fidel castro	Mojito	Fish & Chips
Filippa fizz	Vodka redbull	Coca Cola Zero
P2	Hot shot	Coca Cola
Melleruds pilsner eko	Sol	Shauns Pannkakstallrik
Razz & sprite	Tequila shot	Extra lantpommes
Tom collins		
Rom & cola		

B.2 Example of recommendations- Seller A

Table B.3: Recommendations (rec.) for user u_2 from Seller A, based on all the items bought by user u_2, using WARP with and without inverse propensity weights, respectively.

True interactions	rec. WARP	rec. WARP inv	rec. baseline
Ginger Beer	Husets burgare	Bao buns	Husets lager
Husets lager	Coca cola Zero	Husets burgare	Husets burgare
A ship full of IPA (alk.fritt)	Bao buns	Sweet potato fries	Kruzovice på fat
	Coca cola	Don pedros sommarsallad	Sweet potato fries
	Tropical burger	Tropical burger	Gin Tonic

Table B.4: Recommendations (rec.) for user u_2 from Seller A, based on all the items bought by user u_2, using ALS.

True interactions	rec. ALS	rec. baseline
Ginger Beer	Sjöslagssallad	Husets lager
Husets Lager	Gin Tonic	Husets burgare
A ship full of IPA (alk.fritt)	Dark & stormy	Kruzovice på fat
	Picarones	Bao buns
	Ekologiskt Kaffe	Sweet potato fries

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2021-06-10

EXAMENSARBETE Considerations when Constructing a Food Recommender System with Sparse Data
STUDENTER Axel Falk, Johan Sievert Lindeskog
HANDLEDARE Dennis Medved (LTH)
EXAMINATOR Elin Anna Topp (LTH)

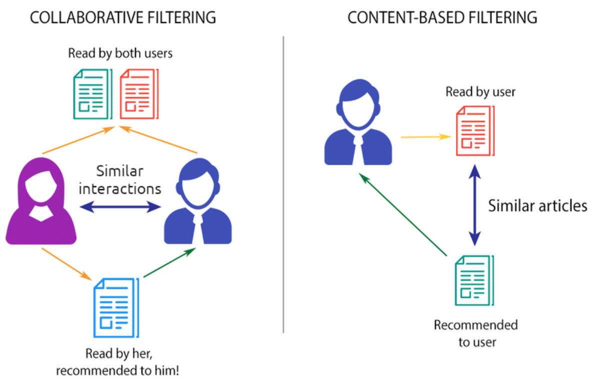
Att tänka på när man implementerar ett rekommendationssystem baserat på gles data

POPULÄRVETENSKAPLIG SAMMANFATTNING **Axel Falk, Johan Sievert Lindeskog**

I snabbt växande ekonomier är företag kopplade till e-handel mer eller mindre beroende av ett bra rekommendationssystem. Systemet kan bidra till både förbättrad användarvänlighet och ökad försäljning. Vi har i detta arbete undersökt möjligheterna att implementera och evaluera ett rekommendationssystem för mat och dryck baserad på ett begränsat, implicit dataset.

De flesta av oss har någon gång stött på någon form av rekommendationssystem, kanske rent av utan att tänka på det. Varje gång man sett klart ett videoklipp på YouTube föreslås nya att se. När man går in på Netflix möts man direkt av en rad rekommendationer, skräddarsydda för just en själv. Lägger man en vara i varukorgen på Amazon.com dyker det upp förslag på vad man också borde köpa till just denna varan. Stora som små bolag kopplade till någon form av e-handel använder idag ofta ett rekommendationssystem för att förbättra användarupplevelsen samt öka sin försäljning.

Detta arbete har genomförts i samarbete med MEIQ Systems AB. Från deras produkt WEIQ, en app för beställning av mat och dryck, har data avseende användare och tillhörande beställningar samlats in. Med hjälp av denna data har vi undersökt möjligheterna att implementera och evaluera ett rekommendationssystem för mat och dryck baserad på begränsad, implicit data. För att kunna göra personliga rekommendationer finns två huvudstrategier: content-based filtering (CBF) och collaborative filtering (CF). CBF utgår från antingen information om användarna, t.ex. demografisk information, eller information om själva artiklarna, t.ex. kategori och taggar. CF utgår istället från preferenser av andra användare. Om två användare har köpt samma artiklar anses de ha liknande preferenser, och man kan därmed göra rekommendationer utifrån detta. Ett exempel av båda typer av rekommendationer finns nedan (Bild från https://www.themarketingtechnologist.co/building-a-recommendation-engine-for-geek-setting-up-the-prerequisites-13).



EXAMENSARBETE Considerations when Constructing a Food Recommender System with Sparse Data
STUDENTER Axel Falk, Johan Sievert Lindeskog
HANDLEDARE Dennis Medved (LTH)
EXAMINATOR Elin Anna Topp (LTH)

Som tidigare nämnt är datan vi jobbat med implicit. Det betyder att användare inte explicit uttryckt sin åsikt avseende en artikel i form av betyg, t.ex. mellan 1 och 5. Istället finns det registrerat vilka användare som köpt vad, vilket ger indikationer till vad en användare tycker om. Då det är enklare att jobba med explicit data, gjordes ett försök till att konvertera den implicita datan till explicit baserat på köpfrekvens, det vill säga att en vara som köpts flera gånger av en användare motsvarar ett högre betyg.

Rekommendationssystem sattes upp för både denna frekvensbaserade explicita datan och den ursprungliga implicita datan. Utvärderingar av rekommendationssystemen visade att rekommendationssystemet konstruerat av den implicita datan var generellt sett bättre, men att även denna inte var helt tillfredställande. Trots att vi provade olika modeller och datauppdelningar, så konstaterade vi slutligen att huvudproblemet var den väldigt begränsade och brusiga datan.