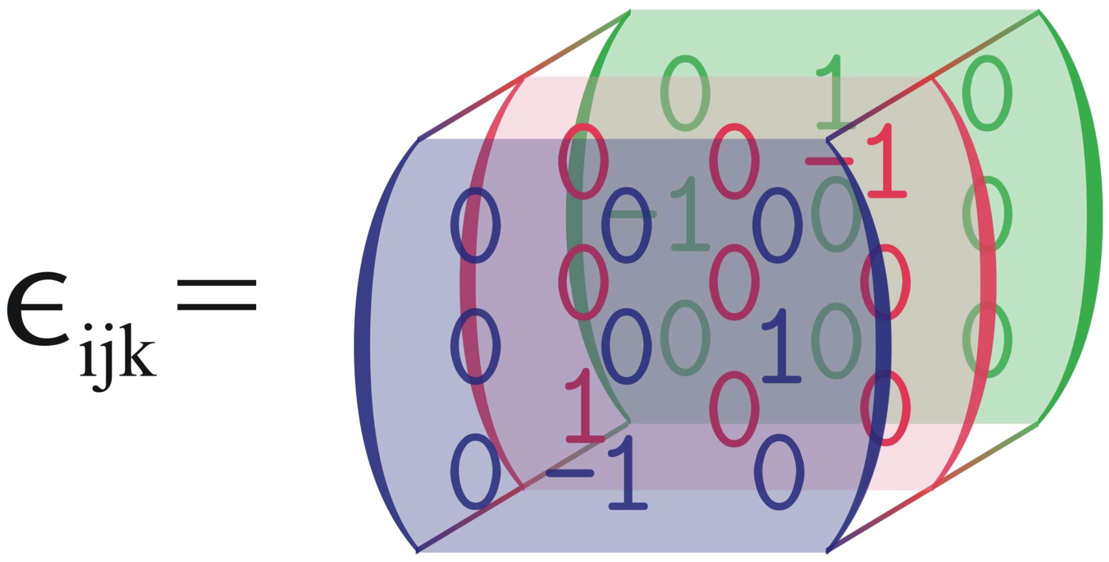
CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG



A hybrid recommender system for usage within e-commerce

Content-boosted, context-aware, and collaborative filtering-based tensor factorization recommender system for targeted advertising within e-commerce.

DATX05

Marcus Lagerstedt
Marcus Olsson

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2017

MASTER'S THESIS 2017

A hybrid recommender system for usage within e-commerce

Content-boosted, context-aware, and collaborative filtering-based tensor factorization recommender system for targeted advertising within e-commerce.

Marcus Lagerstedt
Marcus Olsson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

A hybrid recommender system for usage within e-commerce.

Content-boosted, context-aware, and collaborative filtering-based tensor factorization recommender system for targeted advertising within e-commerce.

Marcus Lagerstedt
Marcus Olsson

Supervisor: Christos Dimitrakakis,
Department of Computer Science and Engineering

Examiner: Richard Johansson,
Department of Computer Science and Engineering

Master's Thesis 2017
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A three-dimensional graphical space used to represent a 3rd order tensor, modelling the idea of user-item-context relationships.

Typeset in LaTeX
Gothenburg, Sweden 2017

A hybrid recommender system for usage within e-commerce.
Content-boosted, context-aware, and collaborative filtering-based tensor factorization recommender system for targeted advertising within e-commerce.

Marcus Lagerstedt
Marcus Olsson

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Recommender systems are information filtering systems that try to predict what rating a user would give an item, usually with the goal of recommending, *would be* high rated items to users. Today there exist recommender systems in most online stores, in one form or another. The complexity of these systems varies greatly, where the less complex ones might base their recommendations on similar products, while others are much more complex, utilizing user modeling etc.

This thesis describes changes made to a context-aware and collaborative filtering-based tensor factorization recommender system, in order to adapt it to perform better with the implicit-only data found in e-commerce, specifically garment-based e-commerce.

Multiple contexts are evaluated in regard to a specific data set, and the performance impact of the changes proposed are also measured. The evaluation is carried out through use of self-implemented algorithms written in Python.

The project resulted in a content-boosted, context-aware, and collaborative filtering-based tensor factorization recommender system made for implicit-only e-commerce data.

The results show that the changes proposed in this thesis give a substantial performance increase, while time-based contexts do not seem to increase performance, in regard to the specific data set used for evaluation in this project.

The best result we got, with the specific data set tested, was about 9.26 % better than that of a recommender system that always recommends the most popular items.

Keywords: recommender system, content-boosted, context-aware, collaborative filtering, tensor factorization, e-commerce, machine learning.

Acknowledgements

We would like to thank the great company Consid AB in general, and our industry supervisor Fredrik Ek in particular, for giving us the opportunity to perform this master thesis with them and for providing us with the data set.

Furthermore we would like to thank our supervisor at Chalmers University of Technology, Christos Dimitrakakis, for the help in finalizing the thesis proposal and for the useful comments and remarks throughout the project.

We would also like to thank our examiner at Chalmers University of Technology, Richard Johansson, for his help and feedback.

Marcus Lagerstedt and Marcus Olsson
Gothenburg, June 2017

Contents

List of figures and plots

List of tables

Notations

Abbreviations

SGD	Stochastic Gradient Descent
ALS	Alternating Least Squares
MPR	Mean Percentage Ranking
MAE	Mean Absolute Error

Variables

$A \circ B$	The element-wise product of matrices A and B
A_i	The ith column of matrix A
K	The number of features to use
D	The number of dimensions of the tensor T
T	A D-dimensional preference tensor that contains only 0:s and 1:s
W	A weight tensor with the same size as T
S_i	The size of T in the ith dimension
$M^{(i)}$	A $K \times S_i$ sized matrix. Its columns are the feature vectors for the entities in the ith dimension
$A_{j_1,\cdots,j_{i-1},j,j_{i+1},\cdots,j_D}$	Denotes an element of tensor A where the index in the ith dimension is fixed to j, and other indices are arbitrary
R	The matrix/tensor of *known*, *real*, ratings
$\hat{R}$	The matrix/tensor of *predicted* ratings

1 Introduction

A *recommender system* is an information filtering system that tries to predict what rating a user would give an item, usually with the goal of recommending *would be* high rated items to users. Lacking research into how well a context-aware tensor factorization-based recommender system work with implicit-only garment-based e-commerce data, there is a need for new research addressing this situation.

1.1 Background

Recommender systems have been around for a long time, and the use of them is more widespread now than ever. Movie recommendations in video-on-demand services, e.g. Netflix, and targeted advertising online, e.g. Google Adwords, are some examples of areas using recommender systems.

Today there exist recommender systems in most online stores, in one form or another. The complexity of these systems varies greatly, where the less complex ones might base their recommendations on similar products or frequently purchased items within the same category as the one being viewed. More complex systems might, however, incorporate factors like the user's purchase history or their demographics.

Research on the topic is still ongoing, with big players such as Google and Netflix [1] doing active research. As an example, recent studies has shown that the performance of recommender systems can be increased using context-aware approaches [2]. Instead of the usual user-item relations, the relations become user-item-context, where *context* may be other information, such as time of rating or the demographics of the users.

The benefits for online stores to deploy a recommender system are many, with targeted advertising resulting in users buying products they would otherwise not have purchased, generating increasing sales, as arguably the most obvious one.

A limitation with the data available in e-commerce is that it often only consists of implicit data in the form of purchase history, compared to most previous studies of tensor factorization-based recommendation algorithms which have been with explicit data, such as user ratings, e.g. 1-10, of items.

1.2 Related work

A popular context-aware tensor factorization-based recommender system is an algorithm proposed by A. Karatzoglou et al. [2]. However, their algorithm can not be used if the data is implicit, e.g. if the ratings are unary.

Another popular recommender system, which works well with implicit data, is an algorithm proposed by Y. Hu et al. [3]. Their recommender system is, however, matrix factorization-based, and does not take context into consideration.

A context-aware tensor factorization-based recommender system algorithm for implicit data is proposed by B. Hidasi et al. [4]. This algorithm is also used as the basis for the work conducted in this project.

1.3 The research question

The research question can be split into two parts: Given the limitations of the implicit-only data available for recommender systems in e-commerce, how well does a tensor factorization-based recommendation algorithm work? Also, how does it compare with the performance of other recommender systems in the same setting, e.g. matrix factorization-based recommender systems, without context-awareness?

1.4 Hypothesis

The hypothesis is that a context-aware recommender system works better than a recommender system without context-awareness. Further, it is expected that the use of available information that does not belong to any specific user or item, i.e. time of purchase, increases the quality of the recommendations. This information cannot be incorporated into the standard form of matrix factorization, thereof the use of tensor factorization.

1.5 Scope

While recommender systems can be used in many areas, this project is delimited to the research of the usage of such systems for targeted advertising within garment-based e-commerce.

The implemented recommender system is not tested in a real live setting, and the evaluation of the performance of the system is limited to offline evaluation using historical data.

1.6 Goals

The goal of this project is to evaluate what performance can be achieved using a content-boosted, context-aware, and collaborative filtering-based tensor factorization recommender system when working with implicit-only e-commerce data.

Another goal of this project is to determine what context information. e.g. time of purchase and age of the user, is relevant and useful, and how this affect the quality of the recommendations.

1.7 Research contribution

There have been numerous studies on collaborative filtering-based recommender systems, such as matrix factorization, using explicit data. However, research using implicit data, such as purchase information, has received less attention. Further, context-aware recommender systems based on tensor factorization has received even less attention, especially regarding implicit data. In the specific case of implicit-only garment-based e-commerce data, there has, to our knowledge, been very little or no research into how well a context-aware tensor factorization-based recommender system work.

We also investigate how the recommender system proposed by B. Hidasi et al. [4] can be extended upon to allow for online updating of the model. The significance of an up-to-date model is also evaluated.

Further, a strategy for minimizing the effect of the cold start problem is proposed and tested.

Finally, how well time-based context information works, in a domain where the lifespan of the products is short, is evaluated.

1.8 Outline of the report

The next chapter will give some background on the field of recommender systems and the challenges of developing such systems. After that, a chapter describing the main research contributions of this thesis will follow. The chapter after that gives information into what experiments have been conducted, as well as how the performance of the developed algorithm was measured. Following that is a chapter presenting the results of the project, followed by a chapter discussing the results. Finally, a conclusion of the project is briefly described and disclosed.

2 Recommender systems

To understand the work that has been conducted during this thesis, it is important to have a general understanding of what a recommender system is, what different categories of recommender systems there are, and the challenges such systems face. This chapter will elaborate on these parts.

2.1 Content-based filtering

Recommender systems using content-based filtering recommend items to a user that are considered similar to what the user is known to have liked in the past. In more detail, all items are described using a set of keywords, which can be assigned manually or automatically by the system. For all users, a profile is constructed indicating what type of items each user likes. These user profiles contain weights for all keywords, denoting the importance of each keyword to the user. The weights are computed by analyzing the items which the user is known to have liked in the past. When recommending items to a user, their user profile is compared to all item descriptions, and the best-matching items are selected [5].

2.2 Collaborative filtering

Recommender systems using collaborative filtering makes use of ratings provided by multiple users, thereof the name *collaborative* filtering. The method works by analyzing a large amount of data on user-item interactions, such as ratings (explicit feedback) or purchases (implicit feedback).

An interesting property of collaborative filtering is that the system does not know the properties of neither the users nor the items; only the interactions between them are known. This property makes the technique a good choice for recommendations of complex items, without obvious descriptive keywords [6].

Collaborative filtering is one of the most widely used recommendation techniques and has been shown to generate high-quality recommendations in many settings. One such setting, relevant to this project, is e-commerce, where Amazon popularized an algorithm for item-to-item collaborative filtering [7].

2.2.1 User-based

The idea with user-based collaborative filtering is to recommend items to a user based on the ratings provided by similar users. The main assumption made is that users that have had similar tastes in the past will continue to have so in the future. As an example, when predicting user u's rating on item i, find users that rated items similarly to user u in the past, that also have rated item i, and use their ratings when predicting user u's rating of item i [8].

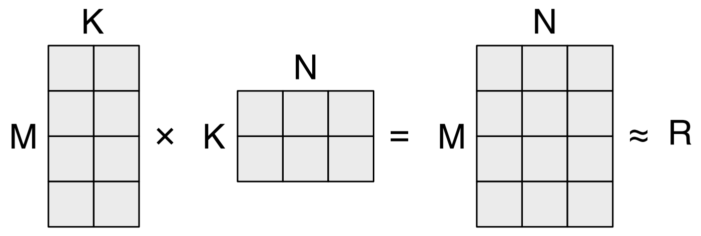
2.2.2 Item-based

With item-based collaborative filtering, the idea is to recommend items to a user that are similar to items previously rated by the user. As an example, given a user u and an item i, collect a set S of items that are both similar to i and that the user u has previously rated. The set S is then used to predict user u's rating of item i [8]. The similarities between the columns in the matrix, which represent the items, are computed to find similarities between items.

2.2.3 Matrix factorization

A widely used method for collaborative filtering is *matrix factorization* [1], [8], [9]. This method uses rank factorization to factorize a user-item matrix of known ratings, $R_{m\times n}$, into low-rank matrices $P_{m\times K}$ and $Q_{n\times K}$, whose product $PQ^T = \hat{R}$ approximates R, where m is the number of users, n is the number of items and K is the rank used for the factorization. Using $\hat{R}$, it is then possible to approximate the missing ratings of R [9].

Figure 2.1: A visual explanation of the low-rank matrices $P_{m\times K}$ and $Q_{n\times K}$ and their product $\hat{R}_{m\times n} \approx R_{m\times n}$, where m is the number of users, n is the number of items, and K is the rank used for the factorization.



In its most basic form, the method tries to simplify both items and users into vectors of length K. A vector $q_i \in \mathbb{R}^K$ is constructed for every item i, and a vector $p_u \in \mathbb{R}^K$ is constructed for every user u. These vectors contain information on the item's features as well as the user's interest in those features. The value of K, the number of features, or *latent factors*, to use, is found from testing different values, with 20-200 often being used. An approximate user rating (e.g. 1-10) for item i by user u is calculated using the dot product of those vectors, $\hat{r}_{ui} = p_u q_i^T$ [9].

Finding the values for the matrices P and Q is done by minimizing the squared error on the set of *known ratings* (R):

$$min \sum_{(u,i)\in R} (r_{ui} - p_u q_i^T)^2 \tag{2.1}$$

To avoid overfitting the observed data, see section 2.5.1, regularizing is used for the learned parameters, whose magnitudes are penalized, making the complete minimization function more like this:

$$min \sum_{(u,i)\in R} (r_{ui} - p_u q_i^T)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2) \tag{2.2}$$

Multiple methods for minimizing the function above exists, with stochastic gradient descent (SGD) and alternating least squares (ALS) being commonly used.

Stochastic gradient descent (SGD)

Stochastic gradient descent, in the setting of matrix factorization, computes the squared errors between the predicted and observed ratings and uses these values when updating the matrices P and Q, aiming to find a minimum of the difference between the ratings.

The error e_{ui} between a predicted rating $\hat{r}_{ui}$ and the observed rating r_{ui}, by user u on item i, is calculated using the following equation [10]:

$$e_{ui}^2 = (r_{ui} - \hat{r}_{ui})^2 = (r_{ui} - \sum_{k=1}^{K} p_{uk} q_{ik})^2 \tag{2.3}$$

where K is the number of features. In order to know in which direction to update the weights p_{uk} and q_{ik}, the gradients of e_{ui}^2 is calculated:

$$\frac{\partial}{\partial p_{uk}} e_{ui}^2 = -2e_{ui}q_{ik} \qquad \frac{\partial}{\partial p_{ik}} e_{ui}^2 = -2e_{ui}p_{uk} \tag{2.4}$$

Using the opposite of the gradients, the update rules for p_{uk} and q_{ik} become [10]:

$$p'_{uk} = p_{uk} + \alpha(2e_{ui}q_{ik}) \qquad q'_{ik} = q_{ik} + \alpha(2e_{ui}p_{uk}) \tag{2.5}$$

where α is the learning rate used. With the usage of regularization, the final update rules look like this:

$$p'_{uk} = p_{uk} + \alpha(2e_{ui}q_{ik} - \lambda p_{uk}) \qquad q'_{ik} = q_{ik} + \alpha(2e_{ui}p_{uk} - \lambda q_{ik}) \tag{2.6}$$

where λ is the regularization value.

Alternating least squares (ALS)

Alternating least squares, in the setting of matrix factorization, finds P and Q by fixing one of them and optimizing on the other. This is done in iterations by assuming that $\hat{P} = P$ and optimizing $\hat{Q}$ to fit $\hat{P}\hat{Q}^T \approx R$, followed by assuming that $\hat{Q} = Q$ and optimizing on $\hat{P}$. This reduces the problem to a problem of linear regression [11].

Computing P by fixing Q. Let $p(u) \in \mathbb{R}^n$ be a vector containing the known binary preferences of user u on the items. Also, denote C^u a matrix of size $n \times n$, where C^u_{ii} denotes the confidence that user u likes item i.

The update rule looks like this:

$$p_u = (Q^T C^u Q + \lambda I)^{-1} Q^T C^u p(u) \tag{2.7}$$

Computing Q by fixing P. Let $p(i) \in \mathbb{R}^m$ be a vector containing the known binary preferences of the users on item i. Also, denote C^i a matrix of size $m \times m$, where C^i_{uu} denotes the confidence that user u likes item i.

The update rule looks like this:

$$q_i = (P^T C^i P + \lambda I)^{-1} P^T C^i p(i) \tag{2.8}$$

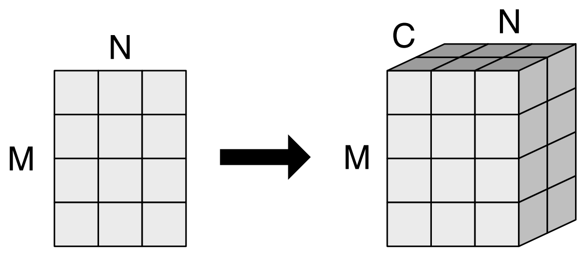
2.3 Context-aware filtering

It has been shown in recent research that the performance of recommender systems can be increased using context-aware approaches [2]. Instead of the usual user-item relations, the relations become user-item-context, where context may be other information such as time of rating or demographics.

2.3.1 Tensor factorization

One way of incorporating such extra information into recommender systems is to generalize the concept of matrix factorization. The two-dimensional matrices in matrix factorization-based algorithms instead become D-dimensional tensors in tensor factorization-based algorithms [2].

Figure 2.2: A visual explanation of the difference of a two-dimensional matrix and a three-dimensional tensor, where M is the number of users, N is the number of items, and C is the number of dimensions of a context.



2.4 Hybrid systems

A hybrid recommender system is a combination of multiple types of machine learning algorithms, e.g. collaborative filtering combined with content- or context-aware filtering, or both of them, in order to create a more robust model [8].

Hybrid recommender systems can be constructed as either a parallel or a sequential construction. In a parallel design, several algorithms would get the same data and produce a result $P_{i,\ i \in \{1..n\}}$, where n is the number of algorithms. All results would in the next step be combined into a single result P. A sequential construction, given a set of algorithms, would instead be: for every ith algorithm, the result $P_{i,\ i \in \{1..n\}}$ is given as input to the next $(i+1)$th algorithm in line [8].

A hybrid recommender system has the potential to make use of the strengths of multiple algorithms, while at the same time minimize the effect of their weaknesses. Collaborative filtering, as an example, with its cold start problem, see section 2.5.4, can be combined with content-based filtering to minimize the cold start problem.

2.5 Challenges

As stated previously, constructing good recommender systems is no easy task. This section describes some of the challenges such systems face.

2.5.1 Data approximation

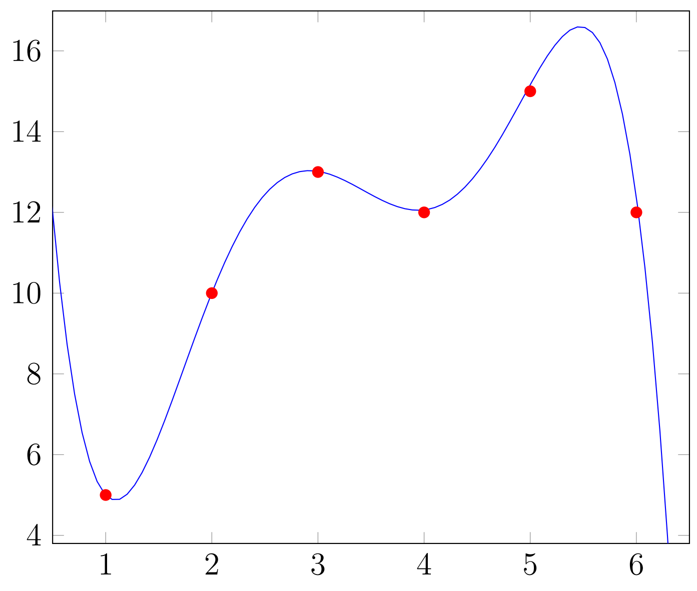
A challenge for recommender systems is *data approximation.* This is a well-known problem in mathematics that also applies here. A summary of the main challenges of data approximation is provided below.

Overfitting

It is easy to assume that any P and Q such that $PQ^T \approx R$ are good, but that is not the case. There exist matrices P and Q that would make the model match the known values of R very closely, but be very bad at predicting the missing values.

This is known as *overfitting* and happens when the model learns details, noise, and randomness in the existing data, as concepts of the model [12]. Overfitting will have good performance on training data, but have poor generalization, resulting in bad predictions. An example of overfitting can be seen in figure 2.3.

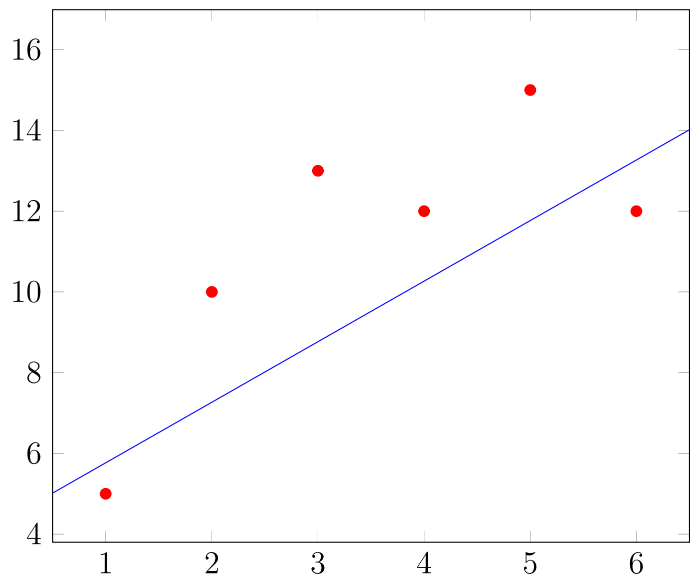
Figure 2.3: Overfitting represented by a function that correctly covers all known points, but at the same time do not cover the general trend.



Underfitting

The opposite to overfitting is called *underfitting*, and it will result in a model that can not represent the training data in a good way nor make good predictions of the missing data. Underfitting happens when the model does not have sufficient accuracy to preserve patterns in the data. A representation of underfitting is shown in figure 2.4.

Figure 2.4: Underfitting represented by a function on training data. The function is far from every point.



Regularization

A way of preventing overfitting is by using *regularization.* In short, regularization works by introducing a penalty for overly complex solutions. The magnitude of the penalty is often determined by a variable denoted λ. In matrix factorization specifically, regularization is used to keep the values of P and Q reasonably small. The goal is to achieve a better generalization of the model, favoring better predictive performance over known ratings matching.

Two commonly used regularization techniques are L_1 and L_2 regularization, and the difference between the two can be described with an example. Let A be a matrix, and b be a vector, both known, in the linear equations problem of finding the unknown vector x so that:

$$Ax = b \tag{2.9}$$

A solution can be found using the following loss function:

$$min \parallel Ax - b \parallel^2 \tag{2.10}$$

In the case of using L_1 regularization, the loss function looks like this:

$$min \parallel Ax - b \parallel^2 + \lambda \parallel x \parallel_1 \tag{2.11}$$

While in the case of using L_2 regularization, the loss function looks like this:

$$min \parallel Ax - b \parallel^2 + \frac{\lambda}{2} \parallel x \parallel_2^2 \tag{2.12}$$

As can be seen, when comparing the two loss functions, the difference is that L_1 uses the sum, or *least absolute deviations*, while L_2 uses the squared sum, or *least squares error.*

Dimension reduction

As previously described in section 2.2.3, matrix factorization tries to simplify both items and users into vectors of length K. This is called *dimension reduction*, and finding an optimal K is important in order to achieve good performance. The goal is to find a value of K large enough to preserve the patterns in the data, while at the same time small enough not to suffer from overfitting [13]. Finding a good value of K is done by trial and error [13].

2.5.2 Implicit feedback

Implicit feedback, like purchases, page views or clicks, is used to indirectly infer a user's preferences.

An overview of the difficulties with working with implicit feedback is provided by Hu et al. [3]. Some of the challenges they describe are listed and described below.

Lack of negative feedback

The challenge with not having explicit data is that with implicit data, an user-item interaction, e.g. a purchase, does not guarantee user satisfaction, and the absence of an user-item interaction does not imply that the user is not interested in the item.

Noise in the data

In an implicit data set, an observed purchase does not guarantee user satisfaction. An item may have been purchased as a gift, or the user may not like what they purchased. These *false positives* may lead a recommender system into learning false patterns, resulting in worse performance.

Confidence vs. preference

In explicit feedback, a high numeric value means that a user gave an item a high rating, indicating a high *preference*, and the opposite for a low numeric value. In implicit feedback though, a high numeric value do not indicate a high preference, but rather, for example, how many times a user purchased an item, instead indicating a high *confidence* that the user likes the item. The number of times a user has purchased an item does not necessarily correspond to how much the user likes the item.

2.5.3 Sparsity

A challenge for matrix- or tensor factorization-based recommender system is that the matrix or tensor to be factorized often is very sparse. As an example, in a system that has 500 000 users, 50 000 items and 5 000 000 known ratings, a user-item rating matrix would be 99.98 % empty, and a tensor with contextual information would be empty to an even larger extent. This poses a difficult problem for a purely mathematical approach to approximate the missing data.

2.5.4 Cold start problem

Recommender systems based on matrix- or tensor factorization in its most basic form suffers from what is known as cold start problems. As only user-item(-context(s)) interactions are used, recommendations for new users are essentially random. This is because the latent factor vector for a not before seen user has not been optimized to match the user's ratings or interactions, as there are none.

Minimizing the effects of the cold start problem are especially important in e-commerce, as it is not uncommon for users to only purchase once, often forcing a recommender system to give recommendations to new users.

2.5.5 Computational performance for incremental updates

Another challenge for recommender systems is trends, items that were once popular may not be popular some time later. This is extra challenging in the setting of e-commerce, were short-lived trends in fashion dictate what people want to, or even can, buy. Therefore, it is important that a recommender system in e-commerce updates its model often, and that it can do so quickly and efficiently, making computational performance an important thing to consider.

2.5.6 Equal items with different names

In systems with a lot of items, it is not uncommon that an item, or very similar items, exist multiple times, but with different names. This may result in a recommender system thinking they are different, which may lead to worse performance.

3 Proposed work

This chapter describes the algorithm developed as a part of this thesis. It starts with a description of the algorithm used as the basis for the work, and then goes over the changes made in order to face some of the challenges described in section 2.5.

3.1 Algorithm

The algorithm proposed in this work is based on an algorithm by B. Hidasi et al. [4], with the addition of using known user features to minimize the effect of the cold start problem and the added possibility of fast online training by partial updates of the model.

The algorithm proposed by B. Hidasi et al. [4] is also known as iTALS, where the i stands for implicit, the T for tensor, and ALS for Alternating Least Squares.

iTALS is a general tensor factorization algorithm that scales linearly with the number of non-zero elements of a tensor and works by using two tensors. The first one, T, is a binary tensor where a cell $T_{u,i,c_1,\cdots}$ is 1 if the user u has at least one interaction with item i while the state of the jth context dimension was c_j, and 0 otherwise. This result in a very sparse tensor, where the proportion of 1:s is very low. The second tensor is called W and contains the weights of all elements in T. The weight of an element t in the tensor T is w_0 if t is 0, and greater than w_0 otherwise. Further, T is decomposed into D matrices, where D is the number of dimensions of T. These matrices are called $M^{(i)}$ and have the size $K \times S_i$, where K is the number of latent factors, S_i is the size of the ith dimension, and $i \in \{1, 2, ..., D\}$.

In this project, T will hold information of observed user-item(-context(s)) interactions, i.e. purchases. W will hold the assigned weights for the same purchases. Also, if user u has bought item i under context c_1 multiple times, the weight of those purchases will stack and be assigned a higher weight when training the model. The possibility to assign individual weights to purchases also provides an easy way of having more recent purchases affect the model more than older ones, this is, however, outside of the scope of this project.

The weights w_0 and w_1 denote the following:

- w_0 is the weight of a non-existing interaction, in practice set to 1.
- w_1 is the weight of an existing interaction, and it is much larger than w_0.

The reasoning for w_1 to be much larger than w_0 is that observed interactions give much more information than unobserved ones.

To avoid numerical instability and overfitting of the model, L_2 regularization is used, using Tikhonov regularization. The regularization value used is denoted λ. See section 2.5.1 for more information regarding regularization in recommender systems.

The model is trained using a number of iterations, or *epochs*. This number is in practice set to 10, as models tend to hardly improve with more epochs than that.

Finding an approximate rating for a user-item-context(s) tuple means finding an approximate value of the corresponding element in the tensor T. This is done using the element-wise product of the relevant columns from the matrices $M^{(i)}$:

$$\hat{T}_{i_1,i_2,\cdots,i_D} = 1^T M^{(1)}_{i_1} \circ M^{(2)}_{i_2} \circ \cdots \circ M^{(D)}_{i_D} \tag{3.1}$$

The loss function to minimize looks like this:

$$L(M^{(1)}, \cdots, M^{(D)}) = \sum_{i_1=1,\cdots,i_D=1}^{S_1,\cdots,S_D} W_{i_1,\cdots,i_D}(\hat{T}_{i_1,\cdots,i_D} - T_{i_1,\cdots,i_D})^2 \tag{3.2}$$

The loss function L is minimized using alternating least squares, that is, all but one of the $M^{(i)}$ matrices are fixed. See section 2.2.3 for more information on alternating least squares.

The pseudocode of the iTALS algorithm, as described in [4], is given in algorithm 1 below.

Algorithm 1: iTALS

```
input  : $T$: a $D$ dimensional $S_1 \times \cdots \times S_D$ sized tensor of zeroes and ones
         $W$: a $D$ dimensional $S_1 \times \cdots \times S_D$ sized tensor containing the weights
         $K$: number of features
         $E$: number of epochs
         $\lambda$: regularization coefficient
Output: $\{M^{(i)}\}_{i=1,\cdots,D}$ $K \times S_i$ sized low rank matrices
procedure iTALS($T, W, K, E, \lambda$):
    for $i = 1, \cdots, D$ do
        $M^{(i)} \leftarrow K \times S_i$ sized random matrix
        $\mathcal{M}^{(i)} \leftarrow M^{(i)}(M^{(i)})^T$
    end for
    for $e = 1, \cdots, E$ do
        for $i = 1, \cdots, D$ do
            $C^{(i)} \leftarrow w_0 \mathcal{M}^{(1)} \circ \cdots \circ \mathcal{M}^{(i-1)} \circ \mathcal{M}^{(i+1)} \circ \cdots \circ \mathcal{M}^{(D)}$
            $O^{(i)} \leftarrow 0$
            for $j = 1, ..., S_i$ do
                $C^{(i,j)} \leftarrow C^{(i)}$
                $O^{(i,j)} \leftarrow O^{(i)}$
                for all $\{t \mid t = T_{j_1,\cdots,j_{i-1},j,j_{i+1},\cdots,j_D}, t \neq 0\}$ do
                    $W_t \leftarrow diag(W_{j_1,\cdots,j_{i-1},j,j_{i+1},\cdots,j_D} - w_0)$
                    $v \leftarrow M^{(1)}_{j_1} \circ \cdots \circ M^{(i-1)}_{j_{i-1}} \circ M^{(i+1)}_{j_{i+1}} \circ \cdots \circ M^{(D)}_{j_D}$
                    $C^{(i,j)} \leftarrow C^{(i,j)} + v W_t v^T$
                    $O^{(i,j)} \leftarrow O^{(i,j)} + W_t v$
                end for
                $M^{(i)}_j \leftarrow (C^{(i,j)} + \lambda I)^{-1} O^{(i,j)}$
            end for
            $\mathcal{M}^{(i)} \leftarrow M^{(i)}(M^{(i)})^T$
        end for
    end for
    return $\{M^{(i)}\}_{i=1,\cdots,D}$
end procedure
```

3.1.1 Online updating of the model

As explained in section 2.5.5, it is necessary for an online recommender system in e-commerce to be able to learn from new data quickly.

The authors of iTALS do, however, not suggest a way of refreshing the model when new interactions are available, besides re-training the whole model on all historical data, together with the new interactions.

A way refreshing the model is by partial updates, only updating the parts of the model that are directly affected by the new interactions.

Consider the new interaction $(u, i, c_1, \cdots)$. Instead of performing optimization steps on the full matrices $M^{(i)}$, optimization steps are performed only on the directly relevant vectors $M_u^{(1)}$, $M_i^{(2)}$, $M_{c_1}^{(3)}$, ..., resulting in an approximate model accounting for the new interaction.

The idea is that the model, from a global perspective, is not changed very much, while the latent factors for user u, item i and context(s) $c_1, \cdots$, are.

A similar idea has been shown to perform well in matrix factorization-based recommender systems [14].

The pseudocode of the online update procedure is given in algorithm 2 below.

Algorithm 2: Online incremental updates for iTALS

```
Input  : $\hat{M}$: Previously learned model
         $T$: a $D$ dimensional $S_1 \times \cdots \times S_D$ sized tensor of zeroes and ones
         $W$: a $D$ dimensional $S_1 \times \cdots \times S_D$ sized tensor containing the weights
         $\lambda$: regularization coefficient
         $interaction$: new interaction ($user$, $item$, $context_1$, $\cdots$)
Output: Refreshed model $\{M^{(i)}\}_{i=1,\cdots,D}$

procedure online_update($\hat{M}$, $T$, $W$, $\lambda$, $interaction$):
    $M \leftarrow \hat{M}$
    $I \leftarrow interaction$
    update $T$ with $I$
    for $i = 1, \cdots, D$ do
        $\mathcal{M}^{(i)} \leftarrow M^{(i)}(M^{(i)})^T$
    end for
    for $i = 1, \cdots, D$ do
        $j \leftarrow I_i$
        $C^{(i,j)} \leftarrow w_0 \mathcal{M}^{(1)} \circ \cdots \circ \mathcal{M}^{(i-1)} \circ \mathcal{M}^{(i+1)} \circ \cdots \circ \mathcal{M}^{(D)}$
        $O^{(i,j)} \leftarrow 0$
        for all $\{t \mid t = T_{j_1,\cdots,j_{i-1},j,j_{i+1},\cdots,j_D}, t \neq 0\}$ do
            $W_t \leftarrow diag(W_{j_1,\cdots,j_{i-1},j,j_{i+1},\cdots,j_D} - w_0)$
            $v \leftarrow M_{j_1}^{(1)} \circ \cdots \circ M_{j_{i-1}}^{(i-1)} \circ M_{j_{i+1}}^{(i+1)} \circ \cdots \circ M_{j_D}^{(D)}$
            $C^{(i,j)} \leftarrow C^{(i,j)} + vW_t v^T$
            $O^{(i,j)} \leftarrow O^{(i,j)} + W_t v$
        end for
        $M_j^{(i)} \leftarrow (C^{(i,j)} + \lambda I)^{-1} O^{(i,j)}$
        $\mathcal{M}^{(i)} \leftarrow M^{(i)}(M^{(i)})^T$
    end for
    return $\{M^{(i)}\}_{i=1,\cdots,D}$
end procedure
```

3.1.2 Minimizing the effect of the cold start problem

As previously explained in section 2.5.4, it is essential for a recommender system in e-commerce to be able to give relevant recommendations to new users. Therefore, a content-boosted pre-processing step before giving recommendations to a new user u is proposed. Instead of using the latent factors of the new user u in the calculations, the average of the latent factors of users considered *similar* to the user u is used. User similarities are based on features that are known about the user(s).

Incorporating the known user features to minimize the effect of the cold start problem is done using the following steps:

1. All users are categorized into a number of different *user categories* based on their features:

   user category(u) = $(feature_1^u, ..., feature_f^u)$

   where $feature_i^u$ denotes the ith feature of user u, and f denotes the total number of features. The number of different user categories is the product of the number of values each feature can take:

   number of different user categories = $\prod_{i=1}^{f}$ cardinality of the ith feature

2. When recommendations for a new user should be made, the best matching user category, which has users with purchase history in it, is found using the Jaccard index, which measures the similarity between two sets, A and B. It is defined as the size of the intersection of the sets, divided by the size of the union of them:

   $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$

   In the calculation of the recommendations for the new user, the average of the user latent factors of the users in that category is then used instead of the latent factors of the new user:

   average of user latent factors of similar users = $\frac{\sum_{v \in V} M_v^{(i)}}{|V|}$

   where $M^{(i)}$ is the latent factor matrix for the users and V is the set of users considered similar to the new user.

The final recommender system is a content-boosted, context-aware, and collaborative filtering-based tensor factorization algorithm for implicit data.

4 Research methodology

This chapter will go into more detail as to what data was used during the project, what experiments was conducted as well as how the results of these were measured and how the system was evaluated.

4.1 Data set

The data set used in this work has been provided by the IT consultant company *Consid AB*[1] and consists of five years of live e-commerce data from one of their clients, *Junkyard*[2].

The data set contains a lot of data regarding each user and their purchases. However, it has no data regarding click-sessions. Also, the data set does not contain any explicit feedback from the users.

The following properties exist on the different entities used:

Table 4.1: Properties of entities in the data set

Entity	**Properties**
Users	year of birth, gender, country, zip code
Items	category, price, gender
Orders	user id, product id, date, time

Only users with all the properties above and less than 100 orders made were extracted, making the size of the data set as follows:

Table 4.2: Quantity of entities in the data set

Entity	**Quantity**
Users	357 696
Items	47 293
Purchases	2 785 390

[1]http://www.consid.se
[2]https://www.junkyard.se

This result in a sparsity of a user-item matrix of ≈ 99.98 %.

4.2 Contexts

Multiple contexts of two main categories, namely *date and time* and *user metadata*, were identified and tested. These are described in more detail below.

4.2.1 Date and time

The main hypothesis for using date and time as contexts is that costumers may tend to buy different products during different seasons, weeks or time of day. This has been shown to be successful in previous research [15]. An example of when the time of day affects consumption is what music plays on the radio; it is common for music on the radio to be more gentle during the morning than during the evening hours.

Half-year

The hypothesis is that customers tend to buy different clothes in either the first or second half of the year. An example of this can be that customers who purchase their winter clothing during the autumn and their summer clothing during the spring.

Description: January-June or July-December
Dimensions of context: 2

Season of year

The hypothesis is that customers tend to buy different clothes during different seasons of the year. It may be that customers buy clothes for the current season or that they purchase clothes for the next season.

Description:

December-February
March-May
June-August
September-November

Dimensions of context: 4

Clothing collections

The hypothesis is that customers tend to buy different clothes during different clothing collection periods. Clothing manufacturers release new clothing collections during specific times of the year, and that could affect what a customer would like to purchase.

Description:

December, January
February, March
April-June
August
September, October
November

Dimensions of context: 6

Month of year

The hypothesis is that customers tend to buy different clothes during different months, some clothes may only be purchased in a specific month and others may be purchased every month.

Description: Month of year
Dimensions of context: 12

Day of week

The hypothesis is that customers tend to buy different clothes during different days of the week. People may, for example, tend to buy shirts before the weekend.

Description: Day of week
Dimensions of context: 7

Time of day

The hypothesis is that customers tend to buy different clothes during different times of the day, or that some items are purchased more often during certain times of the day. E.g. people awake and purchasing clothes at 3 a.m. may be more alike than other people, and also like the same clothes.

Description: 00-06, 06-12, 12-18, 18-24
Dimensions of context: 4

4.2.2 User metadata

In tensor factorization-based recommender systems, every user, item, and context share the same latent factor space. What any given factor represent is unknown, properties of the users are expected to be learned automatically if they give better results.

Some of the factors *may* represent the location, gender and age of the users, and the relevance of the same factors with respect to the items and context. However, learning these things per user basis require a lot of interactions per user, and in e-commerce most users have very few interactions, making learning these things difficult.

The hypothesis is that if known user metadata are included as extra contexts, these things may be learned on a per user metadata basis, as opposed to on a per user basis. The number of interactions per given user metadata is a lot higher than per user, and the performance of the system may reflect this.

Location

The hypothesis is that customers from the same area buy similar items. As an example, people from cities may not buy the same type of clothing as people from more rural areas.

Description: Country and first digit of the zip code
Dimensions of context: 260

Gender

The hypothesis is that men buy men's clothing and women buy women's clothing.

Description: Male or female
Dimensions of context: 2

Age

The hypothesis is that customers of different ages purchases clothing designed for different ages. The age groups are constructed in such a way that each group contains approximately one-third of all users.

Description: Age; < 22, 22-38 or 38+
Dimensions of context: 3

4.3 Evaluation metrics

Evaluation of implicit feedback recommender systems requires appropriate measures. In the traditional setting where a user specifies an item rating using a numeric score, there are clear metrics to measure the success of the predictions, such as the mean squared error between $\hat{R}$ and R. This section describes the evaluation strategy used.

4.3.1 Baseline algorithms

For comparison, two baseline algorithms, described below, are also tested.

Recommend the most popular items

A simple recommender system that always recommends the most popular items has been shown to perform fairly good; people tend to buy the same items that other customers purchases. Therefore this kind of algorithm makes a good benchmark. If an algorithm can outperform this baseline algorithm, it is at least a *somewhat good* algorithm.

A drawback of a system that always recommends the most popular items is that it does not take time into consideration, it does not differentiate the most popular item from last year from the most popular item this year. This can be improved upon using *time decay*, this is, however, outside the scope of this project and is not be tested.

Basic matrix factorization

In order to get insight into how well the extra contexts work, the performance of the system will also be compared to a basic, context-unaware, matrix factorization-based recommender system. See section 2.2.3 for more information.

4.3.2 Historical data

The historical data is, in chronological order, split into 60 parts, with each part spanning about one month of time. The model is then trained on the first 36 of these parts, spanning about three years of time. The evaluation is then conducted on the remaining 24 parts of the data, spanning about two years of time.

The evaluation is conducted on one of the 24 test parts of the data at a time. After the model has been evaluated using one part of the data, the model is trained on the part on which it was just evaluated, after which the same procedure is repeated on the next part of the data. This is repeated until the system has been evaluated on all 24 test parts.

Mean Percentage Ranking (MPR)

Because of the nature of implicit data, a recall based evaluation metric that evaluates a user's satisfaction with an ordered list of recommended items is used, known as Mean Percentage Ranking (MPR) [3].

Denote $rank_{ui}$ the percentage ranking of item i in an ordered list of items recommended for user u.

$$rank_{ui} = location\ of\ item\ i\ in\ recs_u \times \frac{100}{length\ of\ recs_u} \tag{4.1}$$

where $recs_u$ denotes the ordered list of items recommended for user u. A $rank_{ui}$ of 0 indicates that the item i is the most desired by user u, and a $rank_{ui}$ of 100 indicates the opposite. For random predictions, the expected value of $rank_{ui}$ is 50.

The final evaluation score is then the mean of all percentage rankings:

$$\overline{rank} = \frac{\sum_{u,i} rank_{ui}}{number\ of\ recommendations} \tag{4.2}$$

A lower $\overline{rank}$ is more desirable. A $\overline{rank}$ of 0 indicates that the purchased products always were the top recommended, a $\overline{rank}$ of 100 indicates the opposite.

Mean Absolute Error (MAE)

In order to determine the difference in the approximated model from online updating, compared to the model from a full re-training, the metric mean absolute error is used on the matrices $M^{(i)}$.

Consider two arrays of numbers, x and y, both of length n. The mean absolute error is the average difference between each element pair (x_i, y_i) of the two arrays.

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} = \frac{\sum_{i=1}^{n} e_i}{n} \tag{4.3}$$

The equation above can be generalized as follows to measure the *difference* between two multidimensional tensors:

$$MAE = \frac{\sum_{i=1}^{n} \cdots \sum_{j=1}^{m} |y_{i,\cdots,j} - x_{i,\cdots,j}|}{n \times \cdots \times m} = \frac{\sum_{i=1}^{n} \cdots \sum_{j=1}^{m} e_{i,\cdots,j}}{n \times \cdots \times m} \tag{4.4}$$

5
Results

In this chapter, the results of the changes made to the algorithm described in chapter 3 as well as from the experiments of the various contexts described in section 4.2 are presented.

5.1 Algorithm parameters used in all tests

Different values of the parameters used by the algorithm were tested, and the following were found to work best on the data set and were used in all tests:

Table 5.1: Algorithm parameters used in all tests

Parameter description	**Value of parameter**
Regularization value λ	0.045
Number of epochs	10
Number of factors	40
w_0	1
w_1	100

The regularization value λ and the number of factors to use was found using trial and error. The number of epochs to use and the values of w_0 and w_1 were set to the suggestions from the authors of iTALS [4]. Other values were tested, but none were found to perform better.

5.2 Baseline results

For comparison, two baseline algorithms were tested. The first was with a recommender system that always recommends products based on their historical popularity. The second was a run with the main recommender system, without any context or other changes, e.g. minimization of the effect of the cold start problem. Both are described in more detail in section 4.3.1.

The results of these tests were the following:

Table 5.2: Baseline results

Recommender system	**MPR**
Basic matrix factorization	38.51
Recommend the most popular items	21.28

As shown in the table 5.2 above, the recommender system that always recommends the most popular items perform much better than the basic matrix factorization.

5.3 Contexts

In this section, the results of the experiments of the various contexts described in section 4.2 are presented.

5.3.1 Date and time

The following tests were conducted with only one active context, and without other changes, e.g. minimization of the effect of the cold start problem.

Table 5.3: MPR of the different time and date contexts

Context	**MPR**	**Change compared to *no context***
None	38.51	-
Half-year	40.16	4.28 % worse
Season of year	41.31	7.27 % worse
Clothing collections	42.05	9.19 % worse
Month of year	44.18	14.72 % worse
Day of week	40.99	6.44 % worse
Time of day	39.90	3.48 % worse

As can be seen in table 5.3, no date and time context tested gave a better result than the benchmarking test run without context.

Generated data to verify algorithm soundness regarding date and time

In order to make sure that the algorithm could find and make use of date and time-based contexts, if they were present in the data set and relevant for what purchases were made, the system was run on synthetic, generated, data. The tests were also conducted to make sure that the algorithm does not perform considerably worse if the context information is only noise, and not relevant for what purchases were made.

Four different data sets were generated, where purchases depended or did not depend on context, and where similar users did or did not buy similar stuff. In order to have *similar users*, a set of *user types* were introduced, with users of the same user type buying products from the same subset of products.

The data set was generated as follows:

- 100 000 users
- 10 000 products
- 1 000 user types, users of the same user type buy products from the same subset of products
- 1 000 000 orders ≈ 99.90 % sparsity of a user-item matrix

When generating an order:

- Pick a random user.
- Pick a random date, spanning 5 years.

When picking the product of the order:

- **If no context and no user types**: a random product from all products.
- **If no context and user types**: a random product from a subset of all products, subset depends on the users' user type.
- **If context and no user types**: a random product from a subset of all products, subset depends on the context.
- **If context and user types**: a random product from a subset of a subset of all products, the first subset depends on the users' user type, the second depends on the context.

The context used when generating the data was *season of year.*

The results of these tests were the following:

Table 5.4: Generated data, recommender system is not context-aware

	Context	**No context**
User types	0.56	0.6
No user types	50.28	50.01

Table 5.5: Generated data, recommender system is context-aware

	Context	**No context**
User types	0.4	0.6
No user types	38.11	49.98

As can be seen in table 5.4, when the recommender system is not context-aware, the difference in MPR when the data is generated with and without context is very low. In table 5.5, however, when the recommender system is context-aware, the difference in MPR when the data is generated with and without context is substantial (25-35 % increase in performance).

Comparing the values of table 5.4 and 5.5 also shows what the context-aware algorithm does not perform worse if the data set is generated without context.

5.3.2 User metadata

The following tests with different contexts were conducted with only one active context, and without other changes, e.g. minimization of the effect of the cold start problem.

Table 5.6: MPR of the different user metadata contexts

Context	**MPR**	**Change compared to *no context***
None	38.51	-
Location	37.94	1.48 % better
Gender	37.81	1.82 % better
Age	37.01	3.90 % better

As can be seen in table 5.6, all user metadata contexts tested gave better results than the benchmarking test run without context.

5.4 Minimizing the effect of the cold start problem

Tests were conducted to measure the effects of the method for minimizing the effect of the cold start problem described in section 3.1.2.

The results of these tests were the following:

Table 5.7: Comparison of MPR when using, and not using, the fix for the cold start problem

Context	**MPR without the fix**	**MPR with the fix**	**Change**
None	38.51	25.74	33.16 % better
Location	37.94	26.38	30.47 % better
Gender	37.81	25.22	33.30 % better
Age	37.01	24.49	35.45 % better

As can be seen in table 5.7, using the method described in section 3.1.2 to minimize the effect of the cold start problem gives substantially better results for all contexts, compared to when not using it.

Table 5.8: Comparison of MPR when using the fix for the cold start problem

Context	MPR with the fix	Change compared to ***no context*** with the fix
None	25.74	-
Location	26.38	2.49 % worse
Gender	25.22	2.01 % better
Age	24.49	4.86 % better

As can be seen in table 5.8, location as context does not give any improvements in performance, while both gender and age do.

5.5 Combining context and minimizing the effect of the cold start problem

As the contexts *gender* and *age* work best in table 5.8, a test was conducted with those two contexts combined, with the method for minimizing the effect of the cold start problem also activated.

The result of this test was the following:

Table 5.9: MPR when using multiple contexts and the fix for the cold start problem

Context	MPR with the fix	Change compared to ***no context*** with the fix
None	25.74	-
Gender & Age	24.24	5.83 % better

As can be seen if comparing the results of the two context being active one at a time (see table 5.8), and the result of the contexts combined (see table 5.9), the combination of the two contexts outperforms any of the two being active one at a time.

5.6 Online updating of the model

Tests were conducted to measure the effects of training the model at different intervals, or *batch sizes*. The results of these tests are described in the sections below.

5.6.1 Different batch sizes

The quality of the recommendations given by a model trained at different intervals was measured. The test was run without any context, and without other changes, e.g. minimization of the effect of the cold start problem.

The results were as follows:

Table 5.10: MPR of the different batch sizes

Batch size	MPR	Change compared to biggest batch size
46 423	38.51	-
10 000	35.96	6.62 % better
1 000	34.24	11.09 % better
100	31.75	17.55 % better

The batch size of 46 423 is derived from the size of one 1/60 of the data set, that batch size also re-trained the full mode after every batch. The other batch sizes used online updating of the model after every batch and re-trained the full mode after every 46 423th order. As can be seen in table 5.10, training the model more often gives substantially better results, compared to training the model less often.

5.6.2 Computational performance difference

The difference in the computational performance using the online updating method described in section 3.1.1, compared to re-training the whole model, was also measured. The test started out with a model pre-trained on half of the data and no context, and the difference was measured when training on 100 additional orders.

The result of this test was the following:

Table 5.11: Difference of duration per epoch, learning 100 additional orders, comparing online updating with re-training full model[1]

	Duration per epoch
Re-training full model	39 seconds
Online updating of model	5 seconds ($\approx$ 8x faster)

5.6.3 Model approximation difference

In order to determine the difference in the approximated model from online updating, compared to the model from a full re-training, another test was conducted. The test started out with a model pre-trained on half of the data and no context or other changes, e.g. minimization of the effect of the cold start problem. The difference was measured when training on 100 additional orders.

In the tables below $M^{(1)}$ is the latent factor matrix for the products and $M^{(2)}$ is the latent factor matrix for the users.

[1]Measured on an Apple MacBook Pro "Core i5" 2.7 GHz 16 GB 13" Early 2015

Table 5.12: Properties of $M^{(i)}$ in starting model

	$\boldsymbol{M^{(1)}}$	$\boldsymbol{M^{(2)}}$
Min value	-11.294	-0.151
Max value	11.431	0.216
Average value	0.029	0.001

Table 5.13: Properties of $M^{(i)}$ after 100 additional orders, comparing online updating with re-training full model

	Online updating of model		**Re-training full model**	
	$\boldsymbol{M^{(1)}}$	$\boldsymbol{M^{(2)}}$	$\boldsymbol{M^{(1)}}$	$\boldsymbol{M^{(2)}}$
Min value	-11.294 (0 %)	-0.151 (0 %)	-11.257 (-0.33 %)	-0.150 (-0.66 %)
Max value	11.431 (0 %)	0.216 (0 %)	11.355 (-0.66 %)	0.218 (+0.93 %)
Average value	0.029 (0 %)	0.001 (0 %)	0.029 (0 %)	0.001 (0 %)

As can be seen, when comparing tables 5.12 and 5.13, the online update of the model does not change the measured properties of the model. This is in line with the assumption made in section 3.1.1, describing the method for online updating the model. Also, when comparing the tables, differences of the measured properties of the model when re-training the full model can be seen. The average values of $M^{(i)}$ do not change, while the maximum value of $M^{(2)}$ sees an increase of almost one percent.

Table 5.14: Comparison of the different models (A and B), using the mean absolute error as metric

$\boldsymbol{A}$	$\boldsymbol{B}$	**MAE for** $\boldsymbol{M^{(1)}}$	**MAE for** $\boldsymbol{M^{(2)}}$
Starting model	Online updating of model	109.02 $\times 10^{-6}$	57.78 $\times 10^{-6}$
Starting model	Re-training full model	3565.74 $\times 10^{-6}$	0.73 $\times 10^{-6}$
Online updating of model	Re-training full model	3585.70 $\times 10^{-6}$	0.72 $\times 10^{-6}$

When using the mean absolute error as a metric for measuring the difference between the models, the difference between the starting model and the model from online updating of the model also shows only a small difference. The differences are largest when comparing with the model from re-training the full model, with an MAE of about 36 $\times 10^{-4}$ for $M^{(1)}$.

5.7 Online training with content and context

In order to achieve the best possible result, a test was conducted with the best combination of contexts, i.e. gender and age, with the fix for the cold start problem, and with online updating of the model using a batch size of 100.

The result of this test was the following:

Table 5.15: The best possible result and other results for comparison

Context	**Fix for the cold start problem**	**Online updating**	**MPR**
Gender & Age	Yes	Yes, a batch size of 100	19.31
None	Yes	No	25.74
None	No	Yes, a batch size of 100	31.75
None	No	No	38.51

The result is also 9.26 % better than that of the baseline result of a recommender system always recommends the most popular items (21.28).

6 Discussion

In this chapter, the results presented in chapter 5 are discussed and put into the context of the larger field of recommender systems research, together with thoughts regarding possible future work.

6.1 Evaluating the results

In order to ease the discussion of the results regarding the various aspects in this thesis, this section is divided into four parts.

6.1.1 The usage of additional contexts

In this thesis, two distinct categories of contextual information were tested and evaluated, namely *time and date* and *user metadata*. The performance of these can be found in sections 5.3.1 and 5.3.2, respectively.

As stated in section 1.4, the hypothesis was that the usage of time and date as context would increase the quality of the recommendations. This was, however, not backed up by the conducted experiments, as can be seen in table 5.3. In fact, the opposite was observed; the quality of the recommendations got worse when incorporating time and date as context, compared to not using any contexts at all. This was unexpected, as the usage of time and date as context had previously been shown to perform well in other settings [4]. As mentioned in section 2.5.5, it is extra challenging for a recommender system in the setting of e-commerce, were short-lived trends in fashion dictate what people want to, or even can, buy. This might be why the time-based contexts performed poorly in the experiments. Additional tests were also conducted that showed that the recommender system does learn the patterns of time-based contexts if they exist and are relevant, see section 5.3.1.

In addition to testing time-based contexts, contexts based on user metadata were tested. The results of these tests indicate that user metadata-based contexts correlate better to user purchases than time-based ones. All user metadata contexts tested performed better in the tests, compared to when no context information was used, see 5.3.2. The results indicate that customers of the same age, gender, or living in the same location, tend to an extent purchase the same items to a higher degree than customers with more diverse demographics. This agrees with results of previous research.

6.1.2 Tackling the cold start problem

In addition to evaluating the performance of various contexts, a method of minimizing the effect of the cold start problem was proposed, see section 3.1.2. In this project, this is what had the biggest measured impact on the performance of the recommender system. The conducted experiments show improvements of around 30 % when using the proposed method of minimizing the effect of the cold start problem, compared to when not using it, as can be seen in table 5.7. It might be the case that it is especially important to tackle the cold start problem in e-commerce, as it is not uncommon for users to only purchase once, often forcing a recommender system to give recommendations to new users.

6.1.3 Online updating of the model

As explained in section 2.5.5, it is important for an online recommender system in e-commerce to be able to learn from new data quickly. With this as background, a method of performing online updates of the model was proposed in section 3.1.1. In the experiments, a clear correlation between an up-to-date model and an increase in the quality of the recommendations was shown, as can be seen in table 5.10. The hypothesis is that an up-to-date model is able to better pick up on short-lived trends, compared to a more infrequently updated model.

Another aspect of online updating of the model is that it becomes more feasible to keep the model up-to-date. In an experiment, new data was incorporated into the model $\approx$ 80x faster, as can be seen in table 5.11. This speed up is expected to increase with even more data and a bigger model.

As the procedure of online updating of the model is approximate in nature, tests were conducted to measure the difference between an online updated model and a fully trained model. The results of an online update can be seen in tables 5.13 and 5.14. The experiments show that the model, from a global perspective, is not changed very much, while they at the same time show a large effect on the directly affected parts of the model. The results are in line with what was expected and predicted in section 3.1.1, and also agree with those presented in previous research [14].

6.1.4 Other results and thoughts

In order to achieve the best possible result, a test was conducted with the best combination of contexts, i.e. gender and age, with the fix for the cold start problem, and with online updating of the model using a batch size of 100. The result of this test, which can be seen in table 5.15, was the only one what was better than the result of the baseline algorithm that always recommends products based on their historical popularity, see table 5.2. This fact is a strong indication that tensor factorization-based recommender systems are a bad fit for usage in e-commerce.

As an extra note, in the clothing industry, sales and discounts are common, with increased sales on selected products as the result. A recommending system without the knowledge of such extra information is forced to only be *reactive*, while a recommending system with that knowledge has the possibility to take such information into consideration and be more *proactive*. The data set used in this project did not contain such information, and it is expected that, had such information been available, better performance of the recommender system could have been achieved.

6.2 Future work

The best way of evaluating recommender systems is using real online testing, this, however, was not a possibility during this project.

In a setting where items are rated by the users, recommendations may affect *which* items are rated, but not *how* they are rated, which is what is recorded and predicted. In a setting where users buy items, however, recommendations may affect which items the users buy, which is directly what is recorded and predicted.

This means that evaluation of a recommender system in such a setting cannot be done using historical data alone, as there exists no way of affecting what items are bought. Evaluation on historical data only measures how well one predict the past purchases, rather than whether a user is likely to buy an item based on the recommendations provided [16].

We, therefore, propose that future work that follows up on our work extend the evaluation of the recommender system into real online testing or with simulated users. Evaluation with simulated users is not used very much in the research of recommender systems, as the traditional setting does not require it as much as the setting used in this work; it is, however, a recognized evaluation strategy [17].

7 Conclusion

In this thesis, it was investigated how well a context-aware tensor factorization-based recommender system algorithm for implicit data could be made to work in a garment-based e-commerce environment. As can be seen in chapter 5, the results were mixed. Additional contextual information based on the time and date of the purchases does not seem to give an increase in the quality of the recommendations, see section 5.3.1. However, contextual information based on user metadata does seem to increase the quality of the recommendations, see section 5.3.2.

In addition to testing various additional contextual information, two changes to the algorithm proposed by B. Hidasi et al. [4] were made, see section 3.

The first was the added possibility of online updating of the model, see section 3.1.1. This change makes the process of updating the model often with new data much more efficient, see section 5.6.2, and make it more feasible to have the model always being up-to-date. The importance of an up to date model was shown in section 5.6, with 15 % improvements in the performance of the recommender system being observed, compared to a less up to date model, see section 5.6.1.

The second change made was a proposed method for minimizing the effect of the cold start problem, see section 3.1.2. With this fix, the improvement in the quality of the recommendations was measured to over 30 %, see section 5.4.

Note: These results are specifically for the data set used in this thesis, and other data set may give other results.

7.1 The research question and hypothesis

The first part of the research question stated in section 1.3 was *given the limitations of the implicit-only data available for recommender systems in e-commerce, how well does a tensor factorization-based recommendation algorithm work?* The answer we have come up with is that it works *okay.* Our results indicate that collaborative filtering-based recommender systems do not seem to work very well in a setting with implicit-only data, and moreover where the average user has very few interactions with items. However, if the correct contextual information is identified and used, together with the changes proposed in chapter 3 to the algorithm proposed by B. Hidasi et al. [4], a recommender system performing better than an algorithm recommending the most popular items can be archived, as can be seen in table 5.2. This fact, however, is a strong indication that tensor factorization-based recommender systems are a bad fit for usage in e-commerce, at least in the form tested in this thesis.

The second part of the question was *how does it compare with the performance of other recommender systems in the same setting, e.g. matrix factorization-based recommender systems, without context-awareness?* Our hypothesis, laid out in section 1.4, was that it would perform better. The answer to this part is that contextual information based on the time and date of the purchases does not seem to give an increase in the quality of the recommendations, while contextual information based on user metadata does seem to give an increase in the quality of the recommendations.

Again, these results are specifically for the data set used in this thesis, and other data set may give other results.

7.2 Limitations of our research

As stated in section 6.2, evaluation of the recommender system using real online testing was not a possibility during this project, and this should be taken into consideration when analyzing the results of this thesis. The limitation of the single data set used should also be taken into consideration. The algorithm, contexts, and proposed changes evaluated in this project might have performed better or worse if tested with another data set.

Bibliography

[1] C. Gomez-Uribe and N. Hunt, "The netflix recommender system", *ACM Transactions on Management Information Systems*, vol. 6, no. 4, pp. 1–19, 2015.

[2] A. Karatzoglou, X. Amatriain, L. Baltrunas, and N. Oliver, "Multiverse recommendation", *Proceedings of the fourth ACM conference on Recommender systems - RecSys '10*, 2010.

[3] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets", in *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, IEEE, 2008, pp. 263–272.

[4] B. Hidasi and D. Tikk, "Fast ALS-based tensor factorization for context-aware recommendation from implicit feedback", *Machine Learning and Knowledge Discovery in Databases*, pp. 67–82, 2012.

[5] R. Van Meteren and M. Van Someren, "Using content-based filtering for recommendation", *Proceedings of the Machine Learning in the New Information Age: MLnet/ECML2000 Workshop*, pp. 47–56, 2000.

[6] F. Ricci, L. Rokach, and B. Shapira, *Recommender systems handbook*, 1st ed. Springer, 2015, pp. 1–35.

[7] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: Item-to-item collaborative filtering", *IEEE Internet Computing*, vol. 7, no. 1, pp. 76–80, 2003.

[8] C. C. Aggarwal, *Recommender Systems*, 1st ed. Springer International Publishing, 2016.

[9] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems", *Computer*, vol. 42, no. 8, pp. 30–37, 2009. DOI: `10.1109/mc.2009.263`.

[10] G. Takács, I. Pilászy, B. Németh, and D. Tikk, "Matrix factorization and neighbor based algorithms for the netflix prize problem", *Proceedings of the 2008 ACM conference on Recommender systems - RecSys '08*, 2008. DOI: `10.1145/1454008.1454049`.

[11] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets", *2008 Eighth IEEE International Conference on Data Mining*, 2008. DOI: `10.1109/icdm.2008.22`.

[12] T. Dietterich, "Overfitting and undercomputing in machine learning", *ACM Computing Surveys*, vol. 27, no. 3, pp. 326–327, 1995. DOI: `10.1145/212094.212114`.

[13] B. Sarwar, G. Karypis, J. Konstan, J. Riedl, and M. U. M. D. of COMPUTER SCIENCE., *Application of Dimensionality*

Reduction in Recommender System - A Case Study. Defense Technical Information Center, 2000. [Online]. Available: `https://books.google.se/books?id=5kONDQEACAAJ`.

[14] X. He, H. Zhang, M.-Y. Kan, and T.-S. Chua, “Fast matrix factorization for online recommendation with implicit feedback”, *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval - SIGIR ’16*, 2016. DOI: `10.1145/2911451.2911489`.

[15] N. N. Liu, B. Cao, M. Zhao, and Q. Yang, “Adapting neighborhood and matrix factorization models for context aware recommendation”, *Proceedings of the Workshop on Context-Aware Movie Recommendation - CAMRa ’10*, 2010. DOI: `10.1145/1869652.1869653`.

[16] F. Garcin, B. Faltings, O. Donatsch, A. Alazzawi, C. Bruttin, and A. Huber, “Offline and online evaluation of news recommender systems at swissinfo.ch”, *Proceedings of the 8th ACM Conference on Recommender systems - RecSys ’14*, 2014. DOI: `10.1145/2645710.2645745`.

[17] G. Shani and A. Gunawardana, “Evaluating recommender systems”, Tech. Rep., Nov. 2009. [Online]. Available: `https://www.microsoft.com/en-us/research/publication/evaluating-recommender-systems/`.