MŰEGYETEM 1782

Context-aware factorization methods for implicit feedback based recommendation problems

PhD thesis

Balázs Hidasi

Under the supervision of Dr. Gábor Magyar
External advisor: Dr. Domonkos Tikk

Budapest University of Technology and Economics, Hungary
Department of Telecommunications and Mediainformatics
Doctoral School of Informatics

February, 2016

Contents

Acknowledgements **7**

1 Preliminaries **9**
1.1 Recommender systems . . . 9
1.1.1 Recommender algorithm . . . 9
1.1.2 Feedback types . . . 12
1.2 Context-awareness in recommenders . . . 13
1.2.1 Data model for context enriched data . . . 14
1.3 Problem definition . . . 15
1.3.1 Notation . . . 15
1.4 Experimental setup . . . 16
1.4.1 Datasets . . . 16
1.4.2 Context dimensions . . . 16
1.4.3 Metrics of recommendation accuracy . . . 17
1.4.4 Optimization of hyperparameters . . . 19

2 Literature survey **21**
2.1 Factorization on implicit data . . . 21
2.1.1 Implicit matrix factorization algorithms . . . 22
2.2 Context-aware factorization . . . 22
2.2.1 Explicit algorithms . . . 23
2.2.2 Implicit methods . . . 24

3 Initialization of MF **25**
3.1 Initialization of matrix factorization . . . 25
3.2 SimFactor algorithm . . . 27
3.2.1 Similarity-based similarities – Sim^{2}Factor . . . 28
3.2.2 Similarity metrics . . . 30
3.2.3 Composite initialization . . . 32
3.2.4 Complexity . . . 32
3.3 Experiments . . . 33
3.4 Summary . . . 38

4 Context-aware factorization **39**
4.1 Modeling of the implicit task . . . 39
4.2 iTALS . . . 41
4.2.1 Derivation of iTALS . . . 41

4.2.2 Complexity 44
4.2.3 Results 45
4.3 iTALSx 47
4.3.1 Derivation of iTALSx 47
4.3.2 Complexity 51
4.3.3 Results 51
4.4 Comparison of iTALS and iTALSx 52
4.4.1 Accuracy comparison 53
4.4.2 Complexity and training times 56
4.5 Summary 57

5 Speeding up ALS **59**
5.1 Improving training times for practical usefulness 59
5.2 Coordinate descent 60
5.2.1 Complexity 63
5.3 Conjugate gradient 63
5.4 Complexity 65
5.5 Comparison of learning methods 66
5.5.1 Recommendation accuracy 67
5.5.2 Running time 70
5.5.3 Accuracy-running time trade-off 71
5.5.4 Number of inner iterations 72
5.5.5 Convergence of accuracy 73
5.5.6 Size of training data 74
5.6 Summary 76

6 The General Factorization Framework **77**
6.1 Preference modeling easily 77
6.2 Basic GFF 78
6.2.1 Training with ALS(-CG) 80
6.2.2 Complexity of training 83
6.2.3 Special cases 83
6.3 Modeling preferences under context 84
6.3.1 Preference models 84
6.3.2 Results 87
6.3.3 Context-context interactions 90
6.3.4 Training time 90
6.4 Comparison with state-of-the-art algorithms 91
6.4.1 Qualitative comparison with factorization methods 92
6.4.2 Quantitative comparison 93
6.5 Extension – MDM compliant GFF 94
6.5.1 Item metadata as attributes 95
6.5.2 Session information 96
6.5.3 Experimental evaluation 96
6.6 Summary 98

Summary **99**

Application **103**

Future research **105**

Bibliography **106**

Acknowledgements

First and foremost I would like to thank Domonkos Tikk, for his continuous support and invaluable help. He encouraged me to start my PhD studies when I was still indecisive and supported me throughout the whole process. I thank him for his help on improving our papers even if it required staying up late before a submission deadline. I also thank him for finding the funding for my conference related expenses. I thank him for encouraging me when I needed it.

I would like to thank Gábor Magyar for supporting me and helping me with my problems during my time at the university. His help alleviated lot of the uncertainties of the PhD studies and thus made it much smoother.

I would like to thank Gábor Takács for the interesting and inspiring discussions on machine learning and recommendation system research. I would also like to thank him for the useful suggestions during the early years of my research about how to improve. I also thank him for introducing the Python language to me that made some things much easier to do.

I would like to thank István Pilászy for the discussions about algorithms that often turned into very interesting brainstorming sessions. I also thank him for sharing his experience of writing dissertations and the PhD process.

I would like to thank Gravity Research and Development Inc. for funding my conference related expenses, so I didn't have to worry about the financial aspect of publishing my papers to conferences. I also thank them for their flexibility that allowed me to work on my research and on my PhD.

Last, but not least, I would like to thank my family and friends who helped me through the rougher parts of these four years and encouraged me to finish this work.

Chapter 1

Preliminaries

1.1 Recommender systems

With the exponential growth of e-commerce, online services and content, users often find themselves overwhelmed by information. The size of the product catalog of e-commerce sites or the content catalog of a video or a news site is too large to be reviewed by humans. On larger sites, the speed of the growth of the content catalog is so high that it is impossible for a normal user to keep up with it. Even if content/products are filtered to a specific topic/category and search options are available, the time required for reviewing them is still considerable. This phenomenon is called information overload.

Recommender systems are information filtering tools that help users in information overload to find interesting items (products, content, etc). Users get personalized recommendations that contain typically a few items deemed to be of user's interest. The relevance of an item with respect to a user is predicted by recommender algorithms; items with the highest prediction scores are displayed to the user.

In the last few years more and more services started using recommender systems and it is becoming a standard feature in online services. These systems benefit the service provider, because users are more likely to find interesting content/product on the site. Therefore they are more likely to remain on the site longer, purchase more products from the service, spend more and/or return more frequently due to being satisfied with the service. Recommender systems are used on e-commerce, news, classified ad and dating sites, in IPTV platforms and in other services.

1.1.1 Recommender algorithm

The core of the recommender system is the recommender algorithm that ranks the items for the users based on their relevance. Recommender algorithms are usually sorted into five main approaches [58] and hybrid algorithms that are the combination of the pure aproaches:

1. **Content based filtering (CBF):** CBF algorithms use item metadata (e.g. author, genre, etc.). First, the metadata of the items is analyzed using text mining and methods from information retrieval [5]. User profiles are built from the metadata of items the user liked/disliked using machine learning. Preferences are predicted by matching the user profile with item metadata [42]. This usually results

in recommendations (very) similar to the items, the user liked before. The main advantages of the CBF methods that it can handle new items and CBF based recommendations are easy to understand for the end user. Its disadvantages are that it recommends very similar items (low serendipity, unable to surprise) and it is less accurate than collaborative filtering [49].

2. **Collaborative filtering (CF):** CF algorithms use only the user–item interactions (also called events or transactions). The assumption of CF is that two users are similar if they consumed similar items; and two items are similar if they have been consumed by similar users [61]. The advantages of CF methods are that they are traditionally accurate and easy to use because they rely on a single information source (events). However CF algorithms only work well with users and and items who have at least a few events, thus they can not recommend to new users and can not recommend new items. This phenomenon is called the cold-start problem [62].

3. **Demographic:** Recommendations are provided based on the demographic profile of the user [43]. Users are segmented into various groups based on socio-demographic information and recommendations to each group are usually manually selected by marketing experts. The demographic approach is rarely used in modern recommenders in its pure form.

4. **Knowledge-based:** Recommenders of this sort are specific to certain domains or even to certain services. Detailed information about features of the items—such as their usefulness for different user needs and their relation—are collected into a knowledge base. Later this information is used to match items to the needs of the users [9, 57]. Knowledge-based approaches provide higher accuracy compared to other methods at the start of their deployment due to using the domain specific knowledge, however later they fall behind methods equipped with learning capabilities. It is also important to point out that the construction of the domain specific knowledge base is very time consuming and generally can be used for a single recommendation problem (can not be transferred to other domains).

5. **Community based:** Also known as social recommender systems [18]. This approach assumes that users have similar tastes/interests to their friends. With the rapid growth of open online social networks, lots of information is available on social relations between users and their individual interests (e.g. likes). Recommenders of this type propagate information on the social network and recommend items that had been liked by certain subsets of the user's connections. The results of these approach are mixed [20, 44] and their careless usage may result in the users disliking the recommender feature of the service.

6. **Hybrid recommenders:** The combination of two or more from the aforementioned approaches. Hybrid approaches aim using the strengths while overcoming the weaknesses of the combined approaches [10].

CF algorithms are the most accurate amongst the pure approaches in a generic situation, e.g. they are more accurate than CBF methods if sufficient preference data is available [49]. CF algorithms can be classified into memory-based and model-based ones. The former are neighbor methods that make use of item or user rating vectors to define similarity, and they calculate recommendations as a weighted average of similar item or user rating vectors (e.g. [15, 37, 56]). In the last decade, model-based methods gained enhanced popularity, because they were found to be much more accurate in the Netflix Prize [8], a community contest launched in late 2006 that provided the largest explicit benchmark data set (100M ratings) for a long time. Model-based methods build generalized models that intend to capture user preference. The most successful approaches are the latent factor algorithms. These represent each user and item as a feature vector in a K dimensional latent feature space.

Matrix Factorization

The most well-known latent feature based algorithms are matrix factorization (MF) methods (e.g. [7, 34, 50, 60, 66, 67]). Matrix factorization methods organize ratings or preferences into a matrix (R), whose dimensions are the users and the items. If user u rates item i with a rating r then $R_{u,i} = r$. R is a large matrix, but very sparse. A user has interactions with only a small fraction of items, thus the majority of rating data is missing. The missing ratings are either considered to be missing or substituted with a constant value (e.g. zeros or the mean of the ratings).

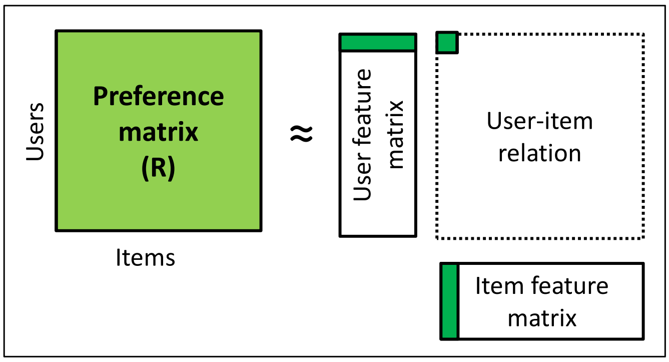

Figure 1.1: Concept of matrix factorization.

The concept behind matrix factorization is to approximate R as the product of two low rank matrices ($\hat{R} = (M^{(I)})^T M^{(U)}$), referred to as feature matrices (see Figure 1.1). One of the feature matrices belongs to the users ($M^{(U)}$) and the other to the items ($M^{(I)}$). The size of the user feature matrix is $S_U \times K$, i.e. it has a row of K length for each user. K is an important hyperparameter of these kinds of algorithms. The u^{th} row of the user feature matrix is the latent feature vector for user u. Latent feature vectors are also assigned to items in a similar fashion. The predicted rating/preference of user u on item i is the dot product of their feature vectors, i.e. $\hat{R}_{u,i} = (M_i^{(I)})^T M_u^{(U)} = \sum_{k=1}^{K} M_{u,k}^{(U)} M_{i,k}^{(I)}$.

The interpretation of matrix factorization is that users and items are projected into a K dimensional space of latent features. Their interaction in this space is used to predict the ratings/preferences. The values in the feature matrices are learned using various optimization procedures that optimize for minimizing the difference between the real and the predicted values of the known coordinates of R w.r.t. a loss function (e.g. root mean squared error (RMSE)).

Some methods (e.g.[53, 64, 65]) do not optimize for the reconstruction of the R

matrix, but rather define other loss functions (e.g. ranking between items) and optimizes the latent features so that they minimize the loss. These methods are latent model based approaches but not matrix factorization methods in the traditional sense. Yet, MF methods and other latent feature based collaborative methods are closely related.

1.1.2 Feedback types

Depending on the nature of the user–item interactions, recommendation problems can be classified into explicit and implicit feedback based problems.

Explicit feedback is provided by the users, usually in the form of ratings, and it explicitly encodes their preferences on the items. In most scenarios, explicit feedback contains both positive and negative feedback. In some special cases either positive or negative feedback is missing [46]. This special case is technically more closely related to the classic implicit feedback problem. As the users interact only with a small fraction of items, the rating data is extremely sparse, in a sense that most of the ratings are missing. The classic explicit feedback based task is rating prediction, where the goal of the algorithm is to accurately estimate missing ratings of the users on items. The accuracy can be measured by classic accuracy metrics such as Root Mean Squared Error (RMSE) or Mean Absolute Error (MAE). This task was set in the focus mostly thanks to the Netflix Prize [8]. The goal of a recommender system however is to present a small number of items to the users that are interesting/useful to them. For this the recommender has to rank the items first (based on their relevance to the given user) and return the first few items on this ranked list. This task is called the topN recommendation. The result of rating prediction can be transformed to topN recommendations by recommending items with the highest predicted ratings for a given user. Good rating prediction not necessarily translates to good topN recommendation.

Implicit feedback is collected via monitoring the behaviour of users while they use a service (e.g. a web shop). User interaction is not required in order to get the feedback, therefore it is available in large quantity. This is of key importance in practical scenarios. Explicit feedback is usually either not available or its amount is negligible compared to implicit feedback. The primary challenge of implicit feedback is that it does not explicitly encode user preferences. These preferences must be inferred from the interactions. The presence of a user action on an item is considered to be a noisy sign of positive preference. For instance, a recommender system may consider the navigation to a particular item page as an implicit sign of preference for the item [46]. The strength of the events' indication of preferences varies on a type by type basis. E.g. purchasing an item is a stronger indicator than looking at a product page (browsing). The assumption of positive preference is not always correct, hence the noise. For instance, a purchased item could be disappointing for the user, so it might not mean a positive feedback.

It is even harder to infer negative feedback as the absence of an event can be traced back to multiple causes, the most common being that the user does not know about the item. Although there are some ways to infer negative feedback in special cases, it is generally assumed to be missing and the absence of positive feedback is considered as a very weak sign of negative preference. Algorithms working with the implicit problem should consider the "missing" feedback in some way.

Although there has been a shift in algorithm research towards the more practical setting of topN recommendations based on implicit feedback in the last few years [30, 64, 65], the majority of research still focuses on ratings and rating prediction.

1.2 Context-awareness in recommenders

Context-aware recommender systems (CARS)[2] consider additional information (termed contextual information or briefly context) besides user–item interactions. Any information can be considered as context, however here I distinguish event context from other types of data, such as item metadata, socio-demographic information or social network of the user. The common property of latter categories that they are bound either to the item or to the user. Also, they are thoroughly examined in specific research topics, such as content based or hybrid recommenders. On the other hand, event context is associated with the interaction of the users and items and can not be bounded to either one. Typical examples are the time or the location of the event. The hypothesis of context-aware recommendations is that they can significantly improve recommendation accuracy, because: (1) Context related effects can be handled during training. For example, certain shifts in the behaviour, like seasonal changes, are only understandable with the proper context provided. For algorithms that do not consider context, these variations seem to be semi-random and can not be handled properly, thus the result will be similar to learning on noisy data. (2) Recommendation lists can be tailored according to the actual value of the context, which may influence the users' needs. For example, a movie that is all about the spectacle will be recommended when the user uses his TV and not when he watches the small screen of his mobile device.

Context can be either explicit, implicit or inferred. Explicit context is provided by the user (e.g. mood). It is very rare that a service provides an option to provide explicit context and even if it is the case, the context data is very scarce. Implicit context is collected by the system and does not require the contribution of users. Typical implicit contexts are the time of the transaction, the location of the transaction and parameters about the device/program through which the user initiated the transaction. Although the general availability of a certain context in a service is high, it is possible that the values are missing for some events. Inferred context is an intermediate category between explicit and implicit context. Explicit context like information is inferred from the behaviour of the user. For example, if the user watches sad movies he must be sad. Due to the elicitation process this type of context is usually unreliable.

Context data can be of various type. It is often assumed that context values are nominal and atomic (e.g. device, mood). Other kind of contexts, such as hierarchical, composite, orderal and continuous contexts also exist. However it is common practice to transform non-nominal context to nominal.

Context-aware recommendation algorithms can be divided into three groups [2]: (1) pre-filtering approaches partition the training data according to the value of the context(s) and train traditional (non context-aware) algorithms on said partitions; (2) post-filtering approaches disregard the context during training, but modify the list of recommendations according to the actual context-state; (3) contextual modeling approaches consider the context dimension(s) during the learning process.

1.2.1 Data model for context enriched data

In this section I briefly review data models for the representation of context-aware data. The focus is on the representation of the input, that is users, items, context; the target attribute (e.g. rating, preference) can be added in a straightforward way. One of the most extensive data models for this task is the Multidimensional Dataspace Model (MDM, [3]). In MDM the dataspace is the Cartesian product of N_D dimensions: $DS = D_1 \times D_2 \times \cdots \times D_{N_D}$. Each dimension is the Cartesian product of one or more attributes: $D_i = A_{i,1} \times A_{i,2} \times \cdots \times A_{i,N_i}$. The data model is very similar to that of relational databases. It is usually also required that the values of an attribute come from a set of atomic and nominal attributes. Therefore continuous variables should be discretized and the order between attribute values is disregarded. The data – usually in the form of transactions – is the subset of every possible combination of the attribute values of all attributes of all dimensions.

An example for representing data in MDM is given as follows. Let $D_1 = U$ be the dimension for users, $D_2 = I$ the dimension for items, and $D_3 = L$ the dimension for locations, thus the dataspace is every possible combination of users, items and locations, i.e. $DS = U \times I \times L$. Let us describe the users by their ID, gender and age; the items by their ID and genres; and the location by the city. Note the following: (1) The data model does not require using the IDs for users/items. However in the classical recommendation scenario the system recommends individual items to individual users. Therefore IDs should be present to distinguish them. If the subject of the recommendation is not an item but one item property, the ID can be omitted. (2) If an item can belong to only one genre, then the item dimension has one attribute that contains this information. If an item has multiple genres then either the combination of genres are the attribute values for a single genre attribute or a binary attribute is required for each genre that contains one if the item belongs to that genre (e.g. IsAction, IsComedy, etc.).

Almost all practically used context-enhanced data can be expressed in a more simple dataspace model, where each dimension consist of exactly one attribute. I refer to this dataspace model as single attribute MDM or SA-MDM. Note that if data is representable in SA-MDM it is also representable in a tensor. The SA-MDM representation is powerful enough for commonly used context dimensions, such as time or location. Even context dimensions that contain more than one attributes can be represented in SA-MDM, but less effectively by just ignoring the grouping of attributes by the dimensions. The main conceptual difference is that interactions between attributes of the same dimension (e.g. item IDs and item genres) cannot be captured. By "converting" all attributes to dimensions we lose the information of this grouping and thus assume extra interactions. This may result in much more interactions (and therefore complexity), especially if multi-valued attributes, like genre or category, is decomposed to many binary attributes. Factorization methods (e.g. [23, 24, 51, 63] usually use SA-MDM.

Another way to simplify MDM is by setting a limit on the number of dimensions as well. Matrix factorization (e.g. [30, 53, 66]) limits the number of dimensions to two (one for users, one for items) and several tensor factorization methods work on only three dimensional data (e.g. [52, 63]).

Another interesting variant of MDM is when the number of dimensions is fixed, but the number of attributes in a dimension is not. Prominent examples using such data model are SVDFeature [13], SVD++ [34] and NSVD1 [47]. They use two fixed dimensions: users and items. SVDFeature sets no restrictions on the number and meaning of attributes for neither the users nor the items. SVD++ requires one of the dimensions to contain a single ID attribute only while the other dimension consists of an ID and several other attributes. Usually the user dimension is restricted to the ID and the additional attributes in this case are binary attributes for all item IDs that are set to one if the user intereacted with the given item. NSVD1 also restricts one of the dimensions to an ID attribute, while the other consists of binary entities of descriptor entities. The descriptor entities are either metadata tokens or users that rated the given item.

1.3 Problem definition

This work focuses on solving the context-aware implicit feedback based recommendation task with factorization and is heavily influenced by the practical considerations. The aim of the research is to integrate context and eventually other types of information (e.g. metadata) into factorization algorithms in order to increase recommendation accuracy for implicit feedback based topN recommendations. Context is defined as event context (associated with the transactions, not with the entities of the transaction). The main metric for recommendation accuracy is recall@20 (see Section 1.4).

1.3.1 Notation

The following notation is used:

$A \circ B$	The Hadamard (elementwise) product of A and B. The operands are of equal size. The result at index $(i, j, \ldots)$ is the product of the elements of A and B at index $(i, j, \ldots)$.
A_i	The i^{th} column of matrix A.
$A_{i_1,i_2,\ldots}$	The $(i_1, i_2, \ldots)$ element of tensor/matrix A.
K	The number of features, parameter of the factorization.
N_D	The number of dimensions of the tensor.
R	A N_D dimensional tensor that contains only zeroes and ones (preference tensor).
$r_{i_1,\ldots,i_{N_D}}$	An element of R at the $(i_1, \ldots, i_{N_D})$ index.
$\mathcal{W}(i_1, \ldots, i_{N_D})$	A weight function that assigns a real value to every cell of R.
S_i	The size of T in the i^{th} dimension $(i = 1, \ldots, D)$.
N^+	The number of ratings (explicit case); non-zero elements in tensor T (implicit case).
$M^{(i)}$	A $K \times S_i$ sized feature matrix. Its columns are the feature vectors for the entities in the i^{th} dimension.
$A_{j_1,\ldots,j_{i-1},j,j_{i+1},\ldots,j_D}$	denotes an element of tensor A where the index in the i^{th} dimension is fixed to j, and other indices are arbitrary.
N_I	Number of inner iterations.
U, I, S, Q, C	Reserved for users, items, seasonality, sequentiality and context respectively. Sometimes used instead of indices when distinguishing objects assigned to dimensions (e.g. S_I: number of items, $M^{(U)}$: user feature matrix).

1.4 Experimental setup

This section gives an overview on the experimental setup I used to evaluate my algorithms.

1.4.1 Datasets

I generally use five genuine implicit feedback data sets to evaluate our algorithm. Three of them are public (LastFM 1K, [11]; TV1, TV2, [14]), the other two are proprietary (Grocery, VoD). The properties of the data sets are summarized in Table 1.1. The column "Multi" shows the average multiplicity of user–item pairs in the training events.[1] The train–test splits are time-based: the first event in the test set is after the last event of the training set. The length of the test period was selected to be at least one day, and depends on the domain and the frequency of events. I used the artists as items in LastFM.

Table 1.1: Main properties of the data sets

Dataset	**Domain**	**Training set**				**Test set**	
		#Users	**#Items**	**#Events**	**Multi**	**#Events**	**Length**
Grocery	E-grocery	24947	16883	6238269	3.0279	56449	1 month
TV1	IPTV	70771	773	544947	1.0000	12296	1 week
TV2	IPTV	449684	3398	2528215	1.0000	21866	1 day
LastFM	Music	992	174091	18908597	21.2715	17941	1 day
VoD	IPTV/VoD	480016	46745	22515406	1.2135	1084297	1 day

1.4.2 Context dimensions

The area of context-aware problems is wide, as any additional information to the user–item interaction can be considered as context. Context dimensions assumed to be event contexts, meaning that their value is not determined solely by the user or the item; rather it is bound to the transaction. E.g. the time of the transaction is an event context, while the genres of the item is not.

Throughout this dissertation I use a general CA setup and use the *time* and *the order of the transactions* to derive context variables that are relevant and thus help improving recommendation accuracy. Implicit feedback data does not typically contain many other event context variables: some contexts, like *mood*, require to be explicitly stated, while others, like *location, device*, are specific to domains. Thus, *seasonality* and *sequentiality* are applied as contexts of the transaction. Both contexts can be derived for any dataset that has timestamp associated with its events. Additionally, the two contexts are rather different and capture different aspects of the data. Their applicability to most datasets makes them ideal subject for the experiments. I use either or both of them for evaluating context-aware algorithms, thus a transaction is a 3- or 4-tuple that contains (1) the user, (2) the item, (3) the time band (based on the timestamp) (4) and/or the previously consumed item by the same user.

[1] This value is 1.0 at TV1 and TV2. This is possibly due to preprocessing by the original authors that removed duplicate events.

Seasonality

Many application areas of recommender systems exhibit the seasonality effect, because periodicity can be observed in many human activities. Therefore seasonal data is an obvious choice for context [39]. First we have to define the length of the season. Within a season we do not expect repetitions in the aggregated behavior of users, but we expect that at the same time offset in different seasons, the aggregated behavior of the users will be similar. The length of the season depends on the data. Once we have this, within seasons we need to create *time bands* (bins) that are the possible context-states. Time bands specify the time resolution of a season, which is also data dependent. We can create time bands with equal or different length. In the final step, events are assigned to time bands according to their time stamp.

For *Grocery* a week was defined as the season and the days of the week as the time bands. The argument here is that people usually do shopping on weekly or biweekly basis and that shopping habits differ on weekends and weekdays. One day was used as season for the other four data sets with 4 hour intervals. Note that one can optimize the lengths and distribution of time bands but this is beyond the scope.

Sequentiality

In some domains, like movies or music, users consume similar items. In other domains, like electronic gadgets or e-commerce in general, they avoid items similar to what they already consumed and look for complementary products. Sequential patterns can be observed on both domain types. Sequentiality as a context dimension was introduced by me in [24] and uses the previously consumed item by the user as a context for the actual item. This information helps in the characterizations of repetitiveness related usage patterns and sequential consumption behavior.

During evaluation sequential context is fixed to the item that was targeted by the last transaction of the user in the training set. Thus no information from the test data is used during the evaluation. The other way (i.e. constantly update the context value based on test events) would be valid as well and would result in better results. Because the test data spans over a short period of time that generally contains a few purchasing sessions for the users, preferences thus can be accurately predicted also from this information.

1.4.3 Metrics of recommendation accuracy

I focus on topN recommendations. For a given user–context configuration setting all items are ranked by their predicted preference ($\hat{r}$). Evaluation metrics are calculated on a test set that does not take part in the training in any form. The relevant items for a user–context configuration (i.e. query) are defined as the items on which the user has events under the given context in the test set. Recommended items are the first N items taken from the ranked list of items generated for the query. Generally, I use $N = 20$; results with $N = 10$ and $N = 5$ usually correlate. It is important to note that I rank all items during evaluation. Although there are other evaluation methodologies in which relevant items are ranked against a small random selection of non-relevant items [34] to reduce the time of evaluation; ranking all items gives more accurate results and it tells more about the actual performance of the recommender.

The most commonly used metrics for evaluating topN recommendations are (with all metrics, at most the first N items of the recommendation lists are considered):

1. **Recall:** The ratio of recommended and relevant items to the relevant items. In other words, it is the proportion of test events that were ranked above in the first N places of their corresponding recommendation list. Recall is useful if the number of recommendations received by a query is expected to be fix and if there is no distinction between the N recommended items. Higher recall is better.

2. **Precision:** The ratio of recommended and relevant items to the recommended items. In other words, it is the fraction of the recommended items that were relevant. Precision is useful if recommendations have some kind of cost associated with them and if there is no distinction between the N recommended items. Note that if the number of recommendations is the same for each query, precision can be computed from recall (and vice versa) by a constant multiplication. Higher precision is better.

3. **F1 score:** The harmonic mean of precision and recall. Higher F1 score is better.

4. **Precision-recall curve:** Values of precision and recall from 1 to N. As the number of recommended items increase, recall increases and precision decreases. The curve is useful for finding a good number of items to recommend and a trade-off between recall and precision if the number of recommended items is not fixed.

5. **AUC (Area Under Curve):** The area under the precision-recall curve from recommending 1 item to recommending N. Higher AUC is better. This metric is also referred to as AUPR (Area Under the Precision-Recall curve) in other fields of machine learning.

6. **MAP (Mean Average Precision):** The mean of average precision of the recommendation lists over queries. The average precision of a list generated for a query (i.e. user–context configuration) is the sum of precisions cut-off at each relevant item in the recommendation list divided by the number of relevant items. Average precision is a sort of weighted recall where lower weights are assigned to relevant and recommended items that are further down the list. MAP is useful if there is some kind of distinction between items of different positions on the recommendation list. Higher MAP is better.

7. **NDCG (Normalized Discounted Cumulative Gain):** The ratio of the discounted cumulative gain and the maximum (ideal) DCG value; averaged over queries. The DCG of a recommendation list is the sum of relevances of relevant and recommended items discounted by a function of their position in the list. For example, $\mathrm{DCG_q} = \sum_{\mathrm{i=1}}^{\mathrm{N}} \left(2^{\mathrm{rel_i}} - 1\right) / \log_2(\mathrm{i} + 1)$, where $\mathrm{rel_i} = 1$ if the i^{th} item in the recommendation list is relevant. The ideal DCG is calculated on a ideal recommendation list where items are ordered according to their actual relevance. NDCG is useful if there is some kind of distinction between items of different positions on the recommendation list. Higher NDCG is better.

8. **MRR (Mean Reciprocal Rank):** The average of the reciprocal rank of the relevant items in their corresponding recommendation lists. An alternative definition averages over queries (instead of events) and assigns the reciprocal rank of

the first relevant item in the recommendation list to the query. MRR is useful if there is some kind of distinction between items of different positions on the recommendation list. Higher MRR is better.

9. **Hitrate:** The ratio of queries for which at least one relevant item was recommended to the number of queries. Hitrate is useful if the number of successful recommendations per user does not matter if there is at least one. Higher hitrate is better.

Accuracy metrics of top-N recommendations usually well correlate. The selection of evaluation metric depends on the recommendation interface as well as the domain. The selection of N depends on the estimation of how many recommendations an average user is exposed to during a session.

My primary evaluation metric is recall@20, defined as the ratio of relevant recommended items and relevant items. The reason for using recall@N is threefold: (1) I found that in live recommender systems recall usually correlates well with click-through rate (CTR), that is, an important online metric for recommendation success. (2) Recall@20 is a good proxy of estimating recommendation accuracy offline for real-world applications (similar finding is available in [40]). (3) Recall is event based. The inclusion of context changes the query set of the test data, therefore the comparison by query based metrics is unfair.

If we have no highlighted items in the recommendations (i.e. all recommended items are equal), then it makes sense to disregard the order of the recommended items. Whether this is true is determined by both the interface and the recommendation logic. For example, if we want to show more items or more diverse itemset to a user during a session while still giving relevant recommendations, we can randomize the top N recommendation and recommend the first K of this random order. This way we can overcome showing users the same K items multiple times and have a higher chance for clicking. The goal of the system is to recommend items that the user likes. The @20 comes from a very average setting of recommending 5 items (from a randomized pool of top 20 items) per page and the user having 4–6 page views in a session. Of course these numbers are highly varied in different applications, but we still think that this is a realistic proxy for a real recommender as it can get.

1.4.4 Optimization of hyperparameters

The hyperparameters of the algorithms, such as regularization coefficients, are optimized on a part of the training data (validation set). Then the algorithms are trained on the whole training data (including the validation set) and recall was measured on the test set. The number of epochs is usually set to 10, because (1) I found that factorization methods converge fairly well in at most 10 epochs; (2) the time of the training should be also considered and 10 epochs is usually a good trade-off between time and accuracy in practical settings. The number of features varies, but is usually set between $20 - 80$, which is a good trade-off between accuracy and training time in practice.

Chapter 2

Literature survey

In this chapter I briefly review the literature on implicit feedback based factorization algorithms and factorization used for context-aware recommendations.

2.1 Factorization on implicit data

The research community of recommender systems traditionally focuses on rating prediction on explicit data. This can be attributed to several influencing factors. (a) Rating prediction is a very well defined problem with a simple objective when compared to top-N recommendations. (b) Recommender system research gained lot of attention due to the Netflix prize, which was a rating prediction task. (c) Most of the publicly available datasets are rating based.

When it comes to the industrial application of recommender systems, rating prediction is a suboptimal solution. First, the generation of recommendation lists require top-N recommendation. Although items can be rated by their predicted ratings, error based metrics (e.g. RMSE) and list based measurements (e.g. recall, MAP) usually do not agree on the ordering of algorithms (e.g. the best rating predictor might be the worst when it comes to ranking). Secondly, explicit feedback is usually not available in large quantities in most practical settings and even if it is available, the quantity of implicit information is usually few magnitudes larger (e.g. even if 80% of the users rate actively, there is no explicit data on 20% of them).

There has been some research on implicit data, but it gained more attention since 2012, when several new methods were introduced. However to this day, most of the research uses explicit data, but usually report ranking results besides the error of the rating prediction.

Explicit feedback based algorithms usually can not be applied directly to implicit feedback. When it comes to factorization it usually requires a loss function that is tailored towards the implicit problem and a well scaling learning procedure. The two main approaches for ranking in the implicit task w.r.t. loss functions are pointwise and pairwise ranking. However, the naive minimization of the objective function in the implicit case is typically expensive, as it scales with the size of the user–item matrix. There are two dominant approaches to overcome this difficulty: (1) the trade-off solution that sacrifices the accuracy to some extent for computational efficiency by sampling the data (usually the missing "negative“ feedback is sampled); (2) the direct minimization of the objective function without sampling by decomposing the calculation to independent parts.

2.1.1 Implicit matrix factorization algorithms

iALS/iMF: The seminal work for implicit data based matrix factorization was proposed by [30]. The method is usually referred to as iALS or iMF. Implicit feedback is modeled by pointwise preferences. If a user has an event on an item, positive preference is assumed and a preference value of 1 is assigned. Otherwise negative preference is assumed with a preference score of zero. However the information in the missing negative feedback is less reliable, therefore weights are assigned to every possible transaction. The weight of missing transactions is constant and significantly lower than that of the positive feedback. The method applies direct optimization, alternating least squares learning and decomposes the derivatives of the objective function to user-dependent and item-independent parts, hence the complexity of a training iteration is reduced to scale *linearly* with the number of events.

BPR: Bayesian Personalized Ranking [53] is a pairwise ranking approach. It is a Bayesian approach that samples negative feedback for every positive feedback, i.e. for every event of the user it samples an item that has no transactions with that user. It is assumed that the user prefers the item on which he has an event over the other one. The model parameters with the highest probability in the aposteriori distribution of the model parameters (given the data) are selected. The optimization is done through stochastic gradient descent. The likelihood is the product of the probabilities of the users preferring the item in their events over an other item. It is assumed that these probabilities are independent. The probability of the user preferring the item of the transaction over the other (given the model parameters) is the function of the difference of the prediction scores of the two items. Since preference can be directly derived from the prediction scores (item with the higher score is preferred), the Heaviside step function would be appropriate here. However its non-continuity would make optimization difficult thus sigmoid is used.

RankSGD: This method was proposed by [31] and it optimizes for error-rate. Error-rate is $ER = \sum_{u=1,i=1}^{S_U,S_I} \left((r_{u,i} - r_{u,j}) - (\hat{r}_{u,i} - \hat{r}_{u,j})\right)^2$. $r_{u,i}$ is the preference of user u on item i and it is 1 if the user has any events on the item and 0 otherwise; $\hat{r}_{u,i}$ is the preference, predicted by the algorithm. Low error-rate means that the difference between the predicted preferences of an item pair is small if the actual preferences are close and large otherwise. The optimization is done via stochastic gradient descent (SGD) by sampling negative feedback for each positive one.

RankALS: This is another method [65] that optimizes for error-rate. But instead of sampling it separates the computations to make the learning efficient w.r.t. training times. It achieves similar results to RankSGD.

CLiMF: Collaborative Less is More Filtering optimizes for a smoothed approximation of the mean reciprocal rank (MRR). Optimization is done via SGD.

2.2 Context-aware factorization

Context-aware recommender systems [3] have emerged as an important research area in the last five years and entire workshops are devoted to this topic on major conferences:

CARS series started in 2009, [1]; CAMRA in 2010, [59]. The application fields of context-aware recommenders include among others: point-of-interest [4], video [68], music [16] and news recommendation [41].

The generalization of matrix factorization for more than two dimensions is tensor factorization (TF). Here context dimension(s) are considered along with the user and item dimensions and the ratings/events are organized into a tensor. For example let us have a set of items, users and ratings (or events) and assume that additional context of the ratings is available (e.g. time of the rating). Having C different contexts, the rating data can be cast into a $N_D = C + 2$ dimensional tensor, T. The first dimension corresponds to users, the second to items and the subsequent C dimensions $[3, \dots, N_D]$ are devoted to contexts. Note that the tensor is very sparse: either most of the ratings are missing (with explicit feedback) or most of the entity combinations (events) are not present in the training data (implicit case). Then a low dimensional representation is created for this tensor. Using more than two dimensions however enables creating several different representations.

2.2.1 Explicit algorithms

The majority of context-aware factorization methods operate on ratings. Some of the most notable algorithms are highlighted here.

Multiverse TF: This method [33] is an efficient sparse HOSVD [36] decomposition of the rating tensor. It decomposes a D dimensional sparse tensor into D (low rank) matrices and a D dimensional (low rank) tensor. If the size of the original tensor is $S_1 \times S_2 \times \cdots \times S_D$ and the number of features is K then the size of the matrices are $S_1 \times K$, $S_2 \times K$, $\dots$, $S_D \times K$ and the size of the tensor is $K \times \cdots \times K$. The authors use gradient descent to learn the model. The complexity of one training iteration scales linearly with the number of ratings (N^+) and cubically with the number of features (K), which is a large improvement compared to the dense HOSVD's $O(K \cdot (S_1 + \cdots + S_D)^D)$.

PITF: Pairwise Interaction Tensor Factorization [17, 52] decomposes a three dimensional rating tensor into three feature matrices. A cell of the tensor is approximated as the sum of the pairwise dot products of the three corresponding feature vectors (one from each feature matrix). In other words the rating of user u on item i under context c is approximated as the sum of the dot product between the user's and the item's, the user's and the context's and the item's and the context's feature vector. SGD is used to learn the feature matrices and it scales linearly with the number of ratings.

Factorization Machines: FM [51, 54, 55] is the generalization of PITF. It approaches the problem from a slightly different angle, but the resulting algorithm can still be considered a tensor factorization method. The authors here do not arrange the data into a tensor, they rather create description matrix for the events, in which a sparse description vector is assigned to each event. The authors refer to this matrix and vectors as feature matrix and vector, but since this term is generally used for the matrix/vector of the latent features, I will refer to them as description matrix and vector to avoid confusion. The rows of the description matrix are the ratings and the columns are entities (attributes, features, etc.) that can be associated to events. This association is done through putting other than zero values in appropriate cells of the description matrix. The value also tells how strong the association is. For example, if the columns of the description matrix correspond to the users and the items, then a ratings descriptor vector is one by the column of the appropriate user and item and zero otherwise. In other words it is a record based structure, where the records are the ratings and the attributes are arbitrary attributes that can be associated with at least one rating (e.g. users, items, context, user/item attributes). The value of the rating is the target variable. FM assigns a latent feature vector to every attribute (column) and it approximates the rating value (target variable) as the sum of all pairwise interactions in the latent feature space between the columns of the attributes, weighted by the product of their assigned values in the description matrix. Since the description matrix is extremely sparse, this computation can be done efficiently. Note that although the way of deriving the model is different, the algorithm could also be represented as a tensor factorization using pairwise model, i.e. the generalization of PITF. The description matrix way of representation is more easy to understand when user/item attributes (e.g. genres) are also used, especially if these attributes are multi-value. The learning can be done via SGD, adaptive SGD, coordinate descent (the authors refer to this as alternating least squares, although it is a special case of ALS) and Markov Chain Monte Carlo (MCMC) optimization. One drawback of FM is that it uses every possible pairwise interaction that can slow down learning if the number of non-zero attribute values is high (i.e. lot of side information is used). It is also not clear why some of these interactions are necessary (e.g. interaction between different metadata values of the same item).

SVDFeature: This method [12, 13] can be considered as a special case of FM, however it can be more practical in may cases. It can use any information that can be associated with either the user or the item, but it can not use event context (information that is associated by both the items and users). It assigns a feature vector to every user and item ID and to every value of user and item attributes. The user feature vector is the linear combination of the corresponding user ID's feature vector and that of the user attribute values that suite the user. Item feature vectors are created similarly. A rating is predicted as the dot product of an item and a user feature vector. This model can also be considered as the sum of pairwise interactions in the latent feature space between user and item attributes. By getting rid of unnecessary interactions, i.e. interactions between user attributes or between item attributes its training time is faster than that of FM. It uses SGD to learn the model parameters.

2.2.2 Implicit methods

Only a few factorization algorithms can handle both implicit feedback and context. TFMAP [63] aims to maximize mean average precision (MAP) through pairwise ranking and sampling. It is suggested by [51] that Factorization Machines [54] can be used for the implicit case with BPR as the objective function, however it is not elaborated and sampling for BPR is not trivial when $D > 2$. A very recent method [45] uses Gaussian processes and can be applied for both explicit and implicit cases.

Chapter 3

Initialization of matrix factorization

In this chapter I examine the importance of the initialization of matrix factorization algorithms. I show that if the usual random or zero initialization is replaced by a similarity based one, the performance of the matrix factorization improves significantly [26, 27]. I propose a matrix factorization based initialization method which integrates additional, possibly external information sources—experiments with context and metadata are performed—to calculate the initial weights in the model. The proposed initialization methodology can be combined with arbitrary implicit feedback MF method (see e.g. [50, 67]). Therefore it is a general concept for the initialization of matrix factorization methods. In addition to that I also propose the SimFactor and Sim^{2}Factor algorithms that further improve the quality of the initial vectors [26, 27].

Additional information—such as context, item metadata or information about the user—can be injected into the matrix factorization via initialization. In this section I show how to inject metadata and context related information. The methods presented here can not be considered to be context-aware algorithms, since they do not tailor the recommendations list according to the value of the context. They however are able to simply include additional information in the recommender algorithm that enhances accuracy and can help in overcoming certain problems of collaborative filtering—such as the item cold-start—without having to switch the recommendation algorithm. The methods are generic, thus they can be used with any matrix factorization algorithm, which I demonstrate on the iALS algorithm [30].

3.1 Initialization of matrix factorization

Most of the MF methods are iterative algorithms that start from a random point: the item and user feature matrices are initialized randomly. After some iterations these methods converge to a local optimum that depends on the starting point. The hypothesis is that an appropriate initialization of feature vectors yields that MF methods will produce more accurate feature vectors and therefore give more accurate predictions.

When investigating the feature vectors of accurate MF models, one can observe that similar items (e.g. items belonging to the same product category, or episodes of a movie series) have similar item feature vectors. This suggest that similarity-based initialization of feature vectors may result in more appropriate models. The calculation of the initial item or user feature vectors should obviously be aligned with the learning algorithm applied.

The first generic initialization method assigns descriptor vectors to the entities (users or items) that characterize them from a certain aspect. The hypothesis is that similar items have similar description vectors. Then these descriptor vectors are compressed together to fit the feature size of the matrix factorization. It is assumed that the relation of similarities (e.g. ratios, order, etc.) between original description vectors will be carried over to the compressed description vectors.

The following data will be used to describe the entities:

1. **Item metadata vectors:** let us consider an indexed set of metadata tags, which contains all the possible tags that occur in item metadata (can be textual or categorical). The item metadata vector contains a non-zero value in the i^{th} position if the i^{th} tag occurs in item's metadata. One can apply various weighting schemes (e.g.: tf-idf) to determine the elements of the vectors.

2. **User/Item event vectors:** a user event vector of S_I length indicates with non-zero values for which item the user has at least one event (analog for items).

3. **User/Item context state vectors:** let us define the set of context states as the possible combination of values of context variables. Here we consider only categorical context variables with finite range. For instance if we take *seasonality* as context, and a season is a week and time bands are days, then we have 7 context states. When more than one context variable is used then the context states are the Descartes-product of individual context values. I.e. if additionally we store in another context variable if the purchase was made online or offline, then we have 14 context states. Then the i^{th} element in the user context state vector is non-zero if the user has at least one event in the i^{th} context state (analog for items).

4. **User/Item context-event vectors:** the user context-event vectors have length $S_C \cdot S_I$; each coordinate represents whether user has events on the given item in the given context state (analog for items).

Remember that most of these vectors are typically very sparse (except the context state vectors if there are only a few context values). Note that in each of the above cases, one has several choices in creating the item/user description vectors from the raw data: vectors may be binary, may contain the frequency, or one may apply normalization or a weighting scheme.

A matrix, D, is assembled from the appropriate input vectors (row-wise), which is referred to as the description of the items (D_I) or users (D_U). For this an arbitrary but single data source is selected from the above options; e.g., one may use the item context state data vectors to form D. In order to make use of the description as initial weights in a matrix factorization method, one should compress them to be compliant with the feature size of the MF model. This can be performed by any dimension reduction techniques like PCA [32], matrix factorization, auto-associative multi layer perceptron [35], etc. These methods minimize the information loss at the compression and simultaneously perform noise reduction.

Here I use two methods for compression. The first is a simple matrix factorization, the weighted ALS, that minimizes the weighted squared error of the predictions by fixing one of the feature matrices and computing the rows of the other by using weighted linear regression. When factorizing item description, only the item feature matrix is kept after the factorization process (analog for user description), which is then readily used as initial feature vectors in the iALS algorithm. Due to the alternating nature of learning in iALS, only one of the feature matrices initialization matters as the other is recomputed right away. The second method is SimFactor, which is explained in the next section.

3.2 SimFactor algorithm

Standard dimension reduction techniques may distort the original system of similarities between the entities. Although dimension reduction techniques tend to maintain similarities between the entities to some degree (similar entities are close, different ones are far away; that is why it can be used for good initializations), the relations between some entities can be quite different. I aim to efficiently approximate the similarity matrix to overcome this issue. One could design a method that keeps this property by starting from the similarity matrix of the users/items. The problem with such an approach is that it requires the precomputation of the entire similarity matrix, which is computationally very inefficient. Further, this solution does not scale well, because the matrix has to be stored in memory for the sake of efficient computation. Even when sparse data structures are used for storing similarities, the calculation of the similarity matrix takes a considerable amount of time, when S_U or S_I is large.

SimFactor is a simple algorithm that compresses the description of the items while preserves the relations between the original similarities as much as possible. This method only works for similarity metrics that can be computed via the scalar product of two (transformed) description vectors. The most commonly used metrics in recommendation systems like cosine similarity, adjusted cosine similarity or Pearson correlation can be computed in this way.

The pseudocode of SimFactor is described in Algorithm 3.2.1 (see also Figure 3.1).

Algorithm 3.2.1 SimFactor

```
Input: $D$ matrix that contains the item or the user description
Output: $F$ matrix that contains the feature vectors of the items or users
procedure SimFactor($D$)
    $D' \leftarrow$ Transform($D$)
    $< X, Y > \leftarrow$ FactorizeMatrix($D'$)
    $Z \leftarrow Y^T Y$
    $< U, \Lambda > \leftarrow$ EigenDecomposition($Z$)
    $F \in \mathbb{R}^{S_{entities} \times K}$
    for $i = 1, \ldots, S_{entities}$ do
        $F_i = \sqrt{\Lambda_{i,i}} X_i U$
    end for
    return $F$
end procedure
```

SimFactor starts with the appropriate transformation of the description matrix (line 1; e.g. ℓ_2-normalization when using cosine similarity, see Section 3.2.2). Next, in line 2, a matrix factorization is applied on the description, but in contrast to ALS, both low rank matrices are kept. For the matrix factorization, arbitrary MF method can be used. Here, I use applied weighted ALS.

The steps performed between lines 3 and 8 are also depicted on Figure 3.1. The matrix of similarities (S) is the product of the transformed description matrix and its transpose ($S = D'D'^T$), while the factor matrices (output of the MF method in line 2) approximate the transformed description matrix ($D' \approx XY^T$). Therefore the similarity matrix can be approximated by $S \approx XY^TYX^T$, where Y^TY is a $K \times K$ symmetric (non-singular) matrix, thus its eigen-decomposition always exists in the following form: $Y^TY = U\Lambda U^T$. U and Λ are $K \times K$ matrices. U contains the eigenvectors, Λ is singular and has the eigenvalues in its diagonal. $\Lambda = \sqrt{\Lambda} \cdot \sqrt{\Lambda}$, where $\sqrt{\Lambda}$ is also diagonal and contains the square roots of the eigenvalues. Now the similarity matrix can be approximated as: $S \approx XU\sqrt{\Lambda}\sqrt{\Lambda}U^TX^T$. Introducing the $S_{entities} \times K$ matrix $F = XU\sqrt{\Lambda}$, this can be rewritten as $S \approx FF^T$.

In F, every row is a feature vector for an entity and the scalar product of the i^{th} and j^{th} rows approximates the similarity between the corresponding entities. This way SimFactor produces low-rank feature vectors that try to preserve the original similarity values. These feature vectors can be used as the initial features in the iALS algorithm.

3.2.1 Similarity-based similarities – Sim^{2}Factor

One can also define the similarity between entities based on how similar they are to other entities. Given the similarity matrix of the entities, similarity measures can be calculated on its rows and these values can be used as entity similarity values. This approach can sometimes seize actual similarities between entities more precisely, because even if the original similarity values are inaccurate, the system of those values is often more consistent.

The computation of the similarity matrix is often not practical or impossible due to time and memory limitations. Fortunately the SimFactor algorithm can be modified in a way that enables the approximation of the similarity based similarity values without the computation of the similarity matrix. This modification is referred to as Sim^{2}Factor. The computational complexity of SimFactor and Sim^{2}Factor are the same. Algorithm 3.2.2 describes the method.

Sim^{2}Factor takes description matrix D as an input like SimFactor. The initial steps are the same: first D is transformed in a way that the scalar product of its rows will result in the desired similarity value. In the second step (line 2) an arbitrary matrix factorization is performed. The resulting factor matrices are transformed into the initial feature vectors (F) between lines 3–10. The key of the algorithm is that the similarity based similarity matrix can be calculated as $S' = SS^T = SS = DD^TDD^T$ (S is symmetric). By approximating $D \approx XY^T$ we have $S' \approx XY^TYX^TXY^TYX^T$. Introducing $Z = Y^TY$ ($K \times K$) that can be computed efficiently we get $S' \approx XZX^TXZX^T$. Using $Z' = ZX^T$ we can write $S' \approx XZ'(Z')^TX^T = XZ''X^T$ (Z'' is symmetric). Then the symmetric $K \times K$ sized Z'' is decomposed into product of two $K \times K$ sized matrices in the same way as in SimFactor ($Z'' = (U\sqrt{\Lambda})(U\sqrt{\Lambda})^T$). The initial feature matrix is $F = X(U\sqrt{\Lambda})$. The steps of the transformation is also shown on Figure 3.2.

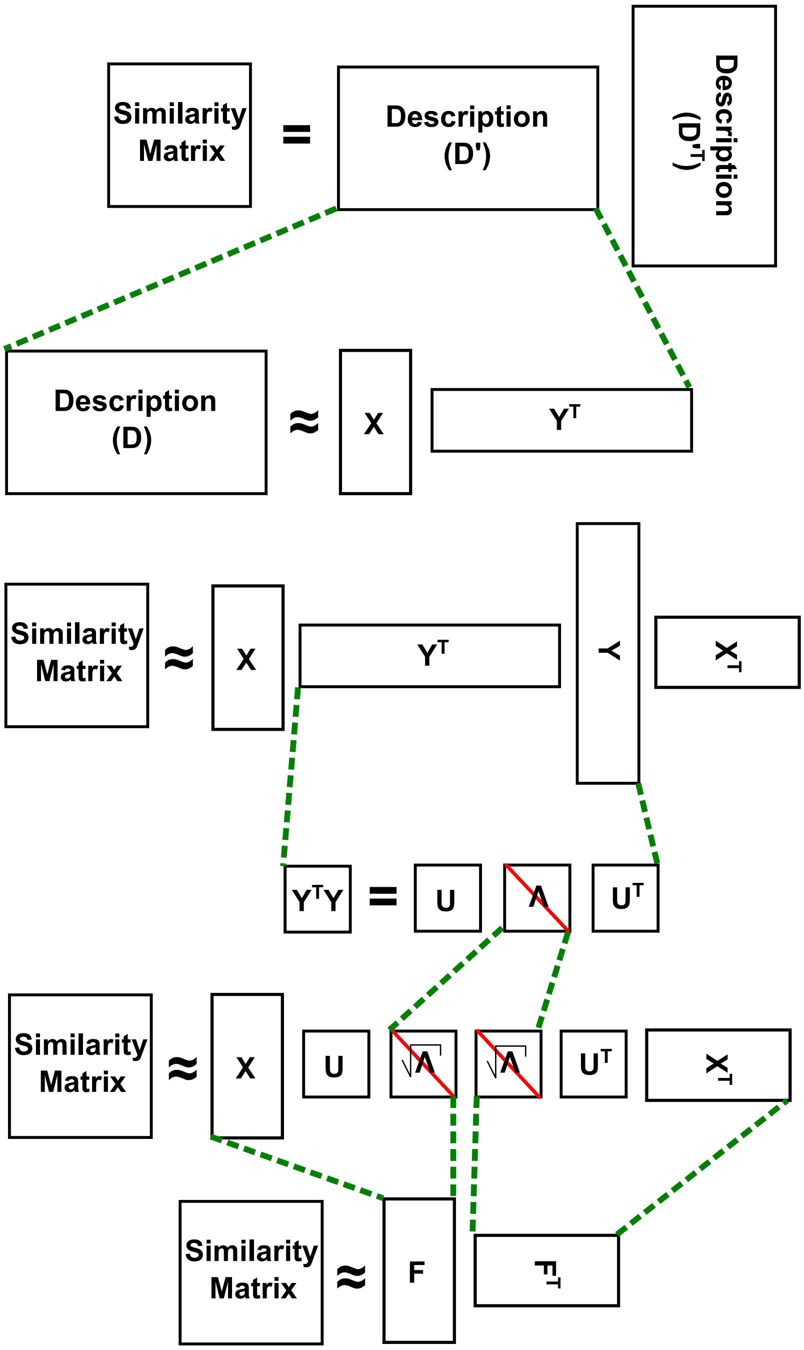

Figure 3.1: Concept of the matrix transformations in SimFactor

Algorithm 3.2.2 Sim^{2}Factor

```
Input: $D$ matrix that contains the item or the user description
Output: $F$ matrix that contains the feature vectors of the items or users
procedure SimFactor($D$)
    $D' \leftarrow$ Transform($D$)
    $< X, Y > \leftarrow$ FactorizeMatrix($D'$)
    $Z \leftarrow Y^T Y$
    $Z' \leftarrow Z X^T$
    $Z'' \leftarrow Z'(Z')^T = (ZX^T)(XZ)$
    $< U, \Lambda > \leftarrow$ EigenDecomposition($Z''$)
    $F \in \mathbb{R}^{S_{entities} \times K}$
    for $i = 1, \ldots, S_{entities}$ do
        $F_i = \sqrt{\Lambda_{i,i}} X_i U$
    end for
    return $F$
end procedure
```

Note that unlike SimFactor, Sim^{2}Factor does not allow the usage of an arbitrary similarity metric between the rows of S. The values of S' will always be the dot products of the rows of S. This is because the description matrix can not be modified efficiently to force the usage of other metrics in S'. However, with the modification of the description matrix, different similarity metrics can be used to approximate S.

3.2.2 Similarity metrics

Different similarity functions can be used to compute the similarity between two descriptor vector and thus get the similarity matrix S. The proper approximation of the similarities with SimFactor requires an initial transformation of the descriptor vectors, so the actual similarity values can be computed as the dot product transformed descriptor vectors. The following similarity functions were used in the experimentation:

1. **Dot product similarity:** The dot product of the corresponding descriptor vectors.

    - *Similarity function:* $s_{\text{dot}}(i, j) = \langle D_i, D_j \rangle$
    - *Transformation:* $D'_i = D_i$
    - *Remarks:* No transformation is required.

2. **Cosine similarity:** The cosine similarity of the corresponding descriptor vectors, i.e. their dot product normalized by their length.

    - *Similarity function:* $s_{\cos}(i, j) = \frac{\langle D_i, D_j \rangle}{\|D_i\|_2 \cdot \|D_j\|_2}$
    - *Transformation:* $D'_i = \frac{D_i}{\|D_i\|_2}$
    - *Remarks:* The descriptor vectors' lengths are normalized to one.

3. **Correlation:** Correlation between two descriptor vectors. Correlation is often better at describing similarities between entities than cosine similarity.

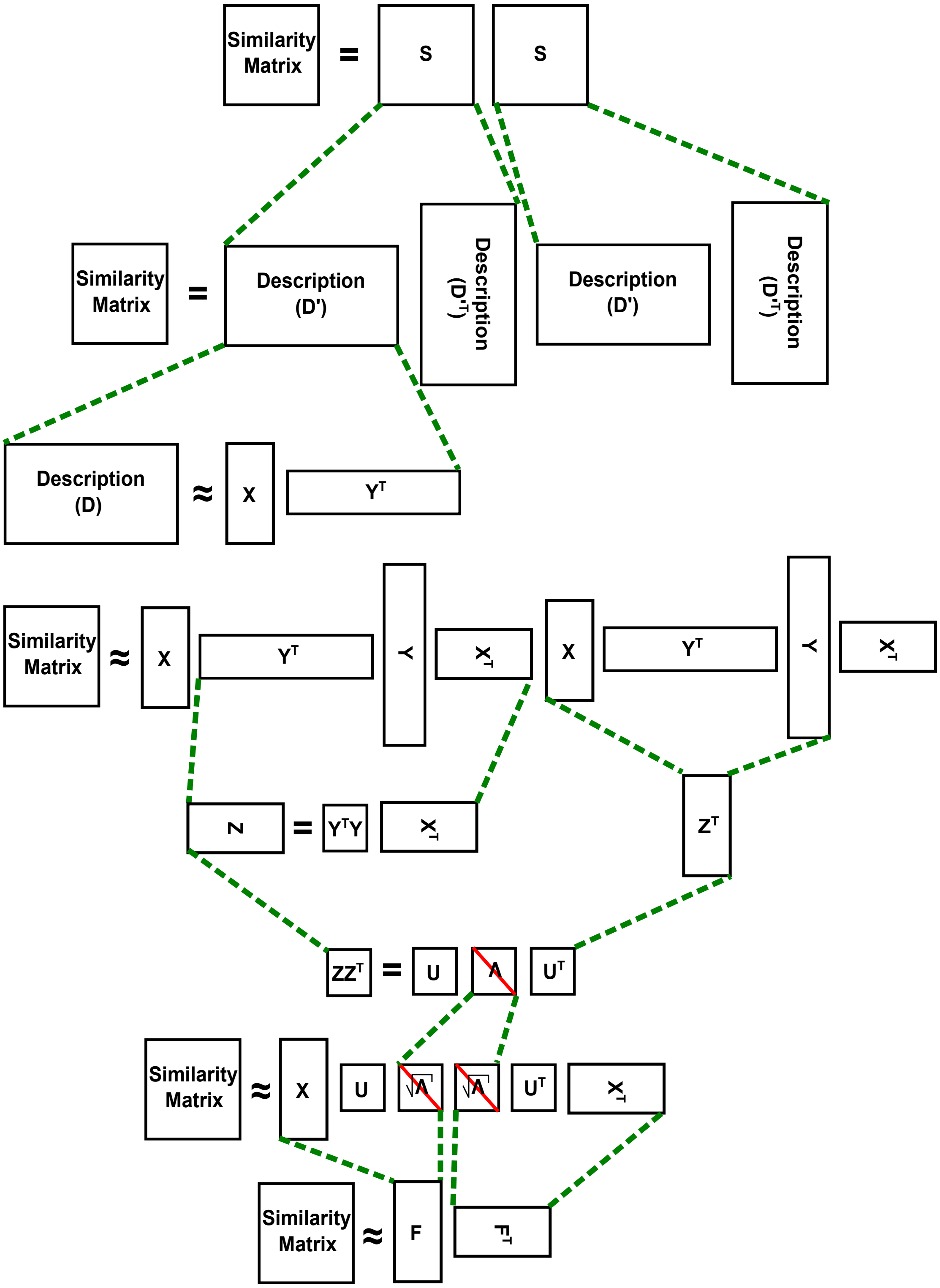

Figure 3.2: Concept of the matrix transformations in Sim^{2}Factor

- *Similarity function:* $s_{\text{corr}}(i,j) = \frac{\left\langle D_i - 1\cdot\frac{\sum_l D_{i,l}}{L}, D_j - 1\cdot\frac{\sum_l D_{j,l}}{L}\right\rangle}{\left\|D_i - 1\cdot\frac{\sum_l D_{i,l}}{L}\right\|_2 \cdot \left\|D_j - 1\cdot\frac{\sum_l D_{j,l}}{L}\right\|_2}$

- *Transformation:* $D'_i = \frac{D_i - 1 \cdot \frac{\sum_l D_{i,l}}{L}}{\left\| D_i - 1 \cdot \frac{\sum_l D_{i,l}}{L} \right\|_2}$
- *Remarks:* This transformation yields a fully specified D' matrix, where the majority of the elements in each row is the same, but their value differ in different rows. Therefore D' can be efficiently stored. The matrix factorization method used in the initialization must also handle the above property of the similarity matrix efficiently. The weighted ALS fulfills this condition.

4. **Binarized correlation:** Correlation matrices can be ill-conditioned e.g. when the values of different rows of the description matrix are from different scale. In such case one may opt for the correlation of the binarized description vectors. For this, D should be first binarized and then the above transformation applied.

   - *Similarity function:* $s_{\text{b.corr}}(i,j) = \frac{\left\langle B_i - 1 \cdot \frac{\sum_l B_{i,l}}{L}, B_j - 1 \cdot \frac{\sum_l B_{j,l}}{L} \right\rangle}{\left\| B_i - 1 \cdot \frac{\sum_l B_{i,l}}{L} \right\|_2 \cdot \left\| B_j - 1 \cdot \frac{\sum_l B_{j,l}}{L} \right\|_2}$, where $B_{i,l} = 1 - I_{D_{i,l}=0}$
   - *Transformation:* $D'_i = \frac{B_i - 1 \cdot \frac{\sum_l B_{i,l}}{L}}{\left\| B_i - 1 \cdot \frac{\sum_l B_{i,l}}{L} \right\|_2}$
   - *Remarks:* First the descriptors are binarized, then the same transformation is applied as with the normal correlation.

3.2.3 Composite initialization

Several possible description matrices were mentioned in Section 3.1 using different information sources. It is possible that by combining these information, higher accuracy can be achieved. There are several ways to combine them.

The first method puts the description matrices one after another, so the description vector of an item is the concatenation of the description vectors ($D = (D_1|D_2|\ldots|D_n)$, here | denotes the concatenation). The method requires matrices to have the same row size that is achieved by imputing rows with zeroes into smaller matrices. Weighting can also be used: $D = (w_1 D_1|w_2 D_2|\ldots|w_n D_n)$ The description matrix created in this manner has many columns and might be hard to factorize precisely.

The second approach defines the initial feature matrix as a weighted combination of the feature matrices created using different descriptions: $F = w_1 F_1 + w_2 F_2 + \ldots + w_n F_n$. The size of all F_i should be the same.

3.2.4 Complexity

Each of the presented initialization methods start with the factorization of the item–descriptor matrix. Since the zeroes in this matrix should be taken into account to get a good approximation of the similarities, using implicit ALS is a natural choice as the factorization algorithm. The cost of this initial factorization is $O(K^3(S_D + S_E) + K^2 D^+)$ where S_D is the number of possible descriptors (e.g.: number of metadata, number of context-states, etc), S_E is the number of entities (e.g. items or users) and D^+ is the number of non-zero values in the descriptor matrix D.

The complexity of SimFactor—in addition to the initial matrix factorization—is $O(K^2 S_D + K^3 + K^2 S_E)$, where the terms correspond to the calculation of $Y^T Y$, finding the eigen-decomposition and calculating $F = XU\sqrt{\Lambda}$, respectively. The cost of Sim^{2}Factor scales similarly to SimFactors (the only difference is in the constant multiplier). This $O(K^2 S_D + K^3 + K^2 S_E)$ cost of the transformation steps is negligible compared to the $O(K^3(S_D + S_E) + K^2 D^+)$ cost of the initial matrix factorization. Therefore the bottleneck of the method is the factorization of the description matrix. If S_D is the same order of magnitude as S_U or S_I then the cost of the initialization is roughly the same as that of the factorization of the user–item matrix therefore the total running time doubles. However this is similar to using a model with either more epochs (iterations) or with slightly higher number of factors and without initialization (and therefore without the additional advantages provided by it). In practice the running time of the implicit ALS is pretty good for low factor models and it can be further enhanced by approximating the ALS algorithm [50, 67]. These enhancements also carry over to the factorization of the description matrix.

3.3 Experiments

Five datasets were used for the experiments. Grocery, TV1, TV2, LastFM (see Section 1.4 for details) and MovieLens 10M [19]. MovieLens is an explicit feedback based dataset, therefore it has to be transformed to simulate implicit feedback. Two separate transformations were done by (1) keeping only the 5 star ratings (referred to as ML5) and (2) keeping ratings with values 4.5 and 5 as positive feedbacks (referred to az ML4.5). We used the 7 days for testing (from 01/12/2008) and the earlier events for training.

Various data sources were used when creating the description matrix (see details in Section 3.1). For contextual information, seasonality was chosen, because the time stamp is available in almost every data set. Different seasons and time band lengths were used and I kept only the best results.

Not all description matrices were used for every dataset. For example sufficient metadata is only available for Grocery. I also found that context-state and context-event descriptors often work similarly, therefore we used the latter only for TV1 and Grocery.

In the first experiment I compared weighted ALS and SimFactor to characterize their similarity preserving capability. I drew randomly 2 times 10 000 entity pairs, calculated the original and the approximated similarity values and counted when the later similarity pairs matched their original order. RMSE of the similarity value prediction was also measured.

The results in Table 3.1 show that SimFactor was more accurate than weighted ALS in every experiment. The improvement in RMSE is usually over 80% except when the description matrix is very sparse. In addition to better accuracy, SimFactor also preserves the original relations of the similarities better than the weighted ALS. One can observe that the results depend greatly on the description matrix.

Besides recall, I used MAP for evaluation. The cut-off was set to 10, 20 and 50. Low-factor models were used as they are of practical use. As baseline, several experiments were run with different random initializations and the best result was chosen. In other

Table 3.1: Accuracy of the similarity prediction

Database	Input data	Non-zero ratio in D	Method	RMSE	RMSE improvement	Order match
Grocery	Item metadata	0.09%	ALS	0.2683	85.51%	67.60%
			SimFactor	0.0389		85.38%
	Item context-state	22.81%	ALS	0.4923	96.09%	92.97%
			SimFactor	0.0192		98.29%
	Item context-event	0.01%	ALS	0.0332	23.47%	63.24%
			SimFactor	0.0254		61.30%
	User context-state	21.97%	ALS	0.3363	99.03%	89.53%
			SimFactor	0.0033		99.92%
	User context-event	0.04%	ALS	0.0425	49.34%	66.70%
			SimFactor	0.0215		74.17%
TV1	Item context-state	66.10%	ALS	0.5056	97.16%	94.32%
			SimFactor	0.0144		97.49%
	Item context-event	0.01%	ALS	0.0602	0.00%	61.57%
			SimFactor	0.0602		61.59%
	User context-state	16.29%	ALS	0.3541	96.78%	87.30%
			SimFactor	0.0114		99.54%
	User context-event	0.04%	ALS	0.1879	20.82%	57.94%
			SimFactor	0.1488		57.51%
TV2	Item context-state	42.02%	ALS	0.4426	94.78%	94.97%
			SimFactor	0.0231		97.84%
	User context-state	5.08%	ALS	0.2669	48.15%	80.00%
			SimFactor	0.1384		80.78%
MovieLens (5)	Item context-state	39.08%	ALS	0.3166	86.59%	90.40%
			SimFactor	0.0425		94.26%
	User context-state	13.53%	ALS	0.3830	99.78%	85.06%
			SimFactor	0.0009		100.00%
MovieLens (4.5)	Item context-state	47.05%	ALS	0.3380	88.02%	91.62%
			SimFactor	0.0405		95.04%
	User context-state	11.74%	ALS	0.3316	75.76%	83.61%
			SimFactor	0.0804		95.44%
LastFM	Item context-state	25.87%	ALS	0.2892	86.87%	84.66%
			SimFactor	0.0380		96.52%
	User context-state	79.69%	ALS	0.5214	97.61%	83.07%
			SimFactor	0.0125		98.75%

words the baseline is the vanilla implicit ALS. I used weighted ALS, SimFactor and Sim^{2}Factor for initialization (the latter two apply weighted ALS as their first step) to create the initial feature vectors. Note that since iALS is an alternating method that discards the results of previous computations when calculating the feature vectors one cannot initialize both item and user features at once as one of them will be discarded in the first step. I ran multiple experiments for each input data type for the initialization and kept only the best for each input data type.

Table 3.2: Recall@50 values for top5 methods per dataset

Dataset	Description	Similarity function	Algorithm	Value	Improvement over baseline
Grocery	User context-state	Cos. Sim.	MF	0.1612	8.40%
	User context-state	Correlation	MF	0.1611	8.33%
	User context-state	Correlation	SimFactor	0.1604	7.85%
	User context-state	Cos. Sim.	SimFactor	0.1602	7.74%
	User context-state	Bin. Corr.	MF	0.1601	7.63%
	Random initialization (baseline)			0.1458	N/A
TV1	User context-event	Cos. Sim.	Sim^{2}Factor	0.2924	7.69%
	User context-event	Bin. Corr.	MF	0.2924	7.69%
	User context-event	Correlation	MF	0.2924	7.69%
	User context-event	Bin. Corr.	Sim^{2}Factor	0.2921	7.57%
	User context-event	Correlation	Sim^{2}Factor	0.2921	7.57%
	Random initialization (baseline)			0.2716	N/A
TV2	User context-state	Cos. Sim.	MF	0.4223	3.73%
	User context-state	Cos. Sim.	SimFactor	0.4210	3.42%
	User context-state	Scalar Prod.	SimFactor	0.4210	3.42%
	User context-state	Correlation	MF	0.4209	3.41%
	User context-state	Bin. Corr.	MF	0.4205	3.29%
	Random initialization (baseline)			0.4071	N/A
ML (5)	User context-state	Bin. Corr.	SimFactor	0.1656	28.57%
	User context-state	Cos. Sim.	SimFactor	0.1626	26.19%
	User context-state	Cos. Sim.	MF	0.1626	26.19%
	Item context-state	Bin. Corr.	Sim^{2}Factor	0.1626	26.19%
	User context-state	Correlation	SimFactor	0.1564	21.43%
	Random initialization (baseline)			0.1288	N/A
ML (4.5)	User context-state	Scalar Prod.	Sim^{2}Factor	0.1334	19.42%
	User context-state	Scalar Prod.	SimFactor	0.1312	17.48%
	User context-state	Cos. Sim.	SimFactor	0.1302	16.50%
	User context-state	Bin. Corr.	Sim^{2}Factor	0.1291	15.53%
	User context-state	Bin. Corr.	SimFactor	0.1291	15.53%
	Random initialization (baseline)			0.1117	N/A
LastFM	User context-state	Bin. Corr.	Sim^{2}Factor	0.0950	113.43%
	Item context-state	Correlation	Sim^{2}Factor	0.0947	112.66%
	Item context-state	Cos. Sim.	Sim^{2}Factor	0.0941	111.25%
	Item context-state	Scalar Prod.	Sim^{2}Factor	0.0941	111.25%
	Item context-state	Scalar Prod.	SimFactor	0.0932	109.34%
	Random initialization (baseline)			0.0445	N/A

Table 3.2 and Table 3.3 summarize the results of our experiments: the top5 initialization methods for each dataset by recall@50 and MAP@50 are shown. Other metrics produce similar results (not shown here). One can observe that all three proposed initialization methods clearly outperform the random initialization. All three methods have similar performance w.r.t. recall, although SimFactor and Sim^{2}Factor seem to have a slight edge over the pure feature based initialization. W.r.t. MAP, the dominance of

Table 3.3: MAP@50 values for top5 methods per dataset

Dataset	Description	Similarity function	Algorithm	Value	Improvement over baseline
Grocery	User context-state	Cos. Sim.	SimFactor	0.1315	8.27%
	User context-state	Cos. Sim.	MF	0.1295	6.65%
	User context-state	Correlation	MF	0.1291	6.29%
	User context-state	Bin. Corr.	MF	0.1283	5.60%
	User context-state	Correlation	SimFactor	0.1275	4.94%
	Random initialization (baseline)			0.1194	N/A
TV1	Item context-event	Bin. Corr.	Sim^{2}Factor	0.0444	24.23%
	Item context-event	Correlation	Sim^{2}Factor	0.0444	24.23%
	Item context-event	Cos. Sim.	Sim^{2}Factor	0.0437	22.40%
	Item context-event	Scalar Prod.	Sim^{2}Factor	0.0437	22.40%
	Item context-state	Cos. Sim.	MF	0.0433	21.36%
	Random initialization (baseline)			0.0357	N/A
TV2	Item context-state	Correlation	SimFactor	0.0770	7.83%
	Item context-state	Cos. Sim.	SimFactor	0.0770	7.82%
	User context-state	Bin. Corr.	SimFactor	0.0764	6.88%
	User context-state	Scalar Prod.	MF	0.0763	6.77%
	User context-state	Cos. Sim.	SimFactor	0.0761	6.50%
	Random initialization (baseline)			0.0714	N/A
ML (5)	User context-state	Cos. Sim.	Sim^{2}Factor	0.0507	19.80%
	User context-state	Scalar Prod.	Sim^{2}Factor	0.0501	18.49%
	User context-state	Bin. Corr.	Sim^{2}Factor	0.0501	18.37%
	User context-state	Bin. Corr.	MF	0.0497	17.50%
	Item context-state	Bin. Corr.	SimFactor	0.0488	15.34%
	Random initialization (baseline)			0.0423	N/A
ML (4.5)	User context-state	Scalar Prod.	SimFactor	0.0333	50.62%
	Item context-state	Scalar Prod.	SimFactor	0.0329	48.81%
	User context-state	Bin. Corr.	Sim^{2}Factor	0.0320	44.70%
	Item context-state	Correlation	Sim^{2}Factor	0.0314	42.09%
	Item context-state	Scalar Prod.	MF	0.0313	41.45%
	Random initialization (baseline)			0.0221	N/A
LastFM	User context-state	Bin. Corr.	MF	0.1474	309.44%
	User context-state	Bin. Corr.	Sim^{2}Factor	0.1369	280.19%
	Item context-state	Correlation	MF	0.1305	262.42%
	Item context-state	Scalar Prod.	Sim^{2}Factor	0.1299	260.74%
	Item context-state	Cos. Sim.	MF	0.1278	255.06%
	Random initialization (baseline)			0.0360	N/A

SimFactor and Sim^{2}Factor is more apparent. Since the additional computations required for these advanced methods is negligible compared to the time of the factorization, it is generally worth to use these. As for the similarity metric, correlation and binarized correlation is often more efficient than cosine similarity.

Figure 3.3 shows the improvement achieved in recall by the top5 methods on each dataset with different cutoffs (10, 20 and 50). The improvement over the baseline is

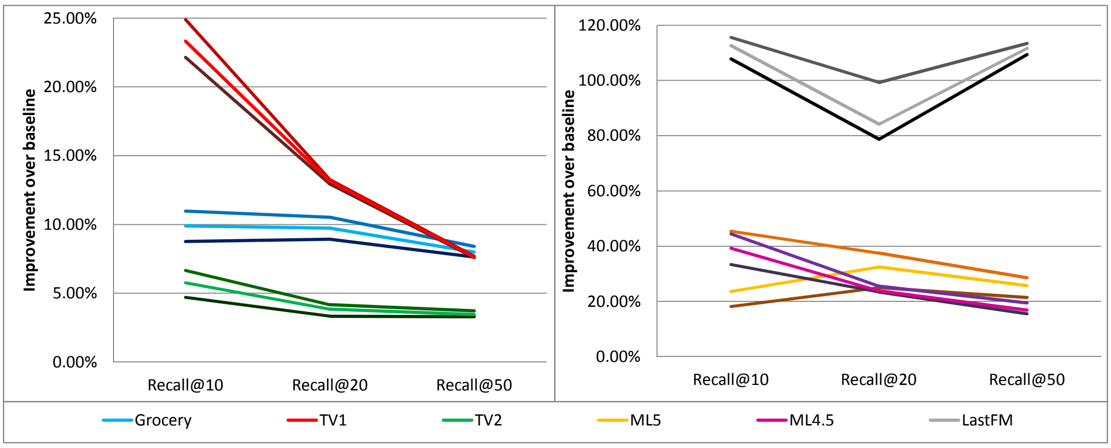

Figure 3.3: Recall improvement over the baseline at cutoffs 10, 20 and 50 for each dataset. Each base color denotes a dataset, the three lines of similar color show the maximal, the average and the minimal improvement by the top5 methods for that dataset.

greater at shorter lists, therefore a proper initialization can be better exploited at the practically more important case.

Note that the top performing methods w.r.t. both evaluation metrics on the Grocery dataset use context for initialization (opposed to metadata). This suggest that context, like seasonality, can efficiently discriminate between entities, and this can be exploited in weight initialization. Users have routines and people with similar routines are similar and might have similar taste. Similarly, different item types are typically consumed in different time bands; for example adult programs mostly viewed late night. The distribution of the events for an entity in the time bands appears to be an efficient descriptor.

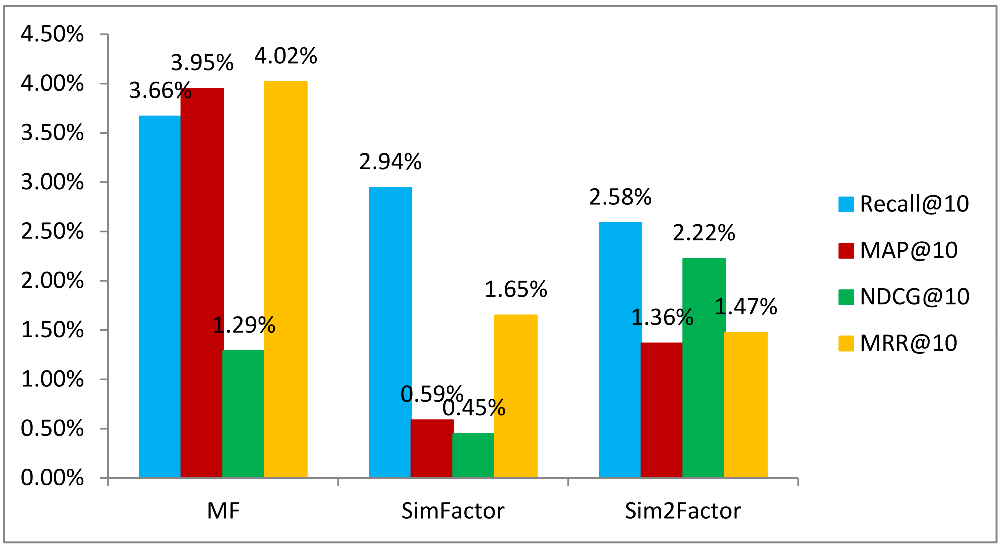

Figure 3.4: Improvement by the composite initialization (over the better of the members; using the same initialization strategy) on short recommendation lists

Experiments with composite initialization were also performed on Grocery, using the context-state and metadata description matrices. When using the concatenation of description matrices, none of the evaluation metric could be improved over the better initialization. On the other hand, using the weighted sum of initial vectors provided better results than both initialization methods. Figure 3.4 shows the improvement by method and metric. As expected, the pure feature based input can benefit more from composite initialization.

3.4 Summary

In Chapter 3 I proposed initializing matrix factorization using information on the items (or users) to increase recommendation accuracy. The methods and the results described in this chapter were published in [26, 27] and the following theses are based on them.

Thesis 1.1 *I proposed to initialize the feature matrices of matrix factorization methods based on the similarities of its entities instead of starting from randomly initialized matrices. The initialization scheme is generic and thus can be applied to any matrix factorization. It consists of two steps: (1) descriptor vectors are assigned to the entities; (2) the descriptors are compressed to fit the size of the feature vectors. I applied the scheme on implicit ALS and showed on five datasets that this type of initialization can increase the recommendation accuracy measured by recall and MAP.*

Thesis 1.2 *I proposed the SimFactor algorithm that yields feature vectors, which preserve the original similarities between entities more accurately. SimFactor does not require the computation of the similarity matrix (which would be infeasible). I showed on five datasets that similarities are better estimated with this algorithm as with pure compression of the descriptor vectors. I also showed that feature vectors yielded by SimFactor are generally better for initializations than those produced by pure compression.*

Thesis 1.3 *I proposed the* Sim^2*Factor algorithm that is able to yield feature vectors whose similarity approximates the similarity between entities, based on how similar they are to the rest of the entities.* Sim^2*Factor does not require the computation of the similarity matrix. I showed that feature vectors of this kind are useful for initialization.*

Thesis 1.4 *I proposed to use context for describing entities. I showed that context based descriptors are better for initialization than metadata based ones. I also showed that the weighted combination of context and metadata based initializations can further improve the recommendation accuracy.*

Chapter 4

Context-aware factorization

In this chapter I introduce two context-aware factorization algorithms for the implicit feedback based recommendation problem, named iTALS [24] and iTALSx [22, 23].

4.1 Modeling of the implicit task

Both algorithms assume data in SA-MDM (see 1.2.1), i.e. data that is representable in an N_D dimensional tensor R. One dimension of the tensor corresponds to the users (user IDs), one to the items (item IDs), while the other $N_D - 2$ dimension is associated with different context dimensions. R contains only zeroes and ones[1]. Let a given element of the tensor be $r_{u,i,c_1,\cdots,c_{N_D-2}} = 1$ if user u has (at least one) event on item i while the context-state of j^{th} context dimension was c_j. Due to its construction, all elements of R are known (i.e. there are no missing "ratings"), however the proportion of ones is very low. This construction of the preference tensor basically assumes that the presence of an event signals positive preference and the absence of an event (i.e. missing feedback) is a sign of negative preference.

Since the missing feedback is clearly a weaker signal of negative preference than the presence of positive feedback (see Section 1.1.2) I construct the $\mathcal{W}(i_1, \dots, i_{N_D})$ weight function that assigns a real value to every possible entity combination. In practice, the construction of $\mathcal{W}(\cdot)$ depends on the problem, and can also affect the complexity of the training. In order to be able to train the model efficiently, $\mathcal{W}(\cdot)$ is restricted as follows:

$$\mathcal{W} : (i_1, \dots, i_{N_D}) \to \mathbb{R}$$
$$\mathcal{W}(i_1, \dots, i_{N_D}) = \begin{cases} w^1(i_1, \dots, i_{N_D}) \gg w^0(i_1, \dots, i_{N_D}), & \text{if } r_{i_1,\dots,i_{N_D}} = 1 \\ w^0(i_1, \dots, i_{N_D}) = \prod_{j=1}^{N_D} \left(\mu^{(j)} v_{i_j}^{(j)} + \gamma^{(j)}\right), & \text{otherwise} \end{cases} \tag{4.1}$$

Where $w^1(i_1, \dots, i_{N_D})$ is the weight of entity combinations of the training set and $w^0(i_1, \dots, i_{N_D})$ is the weight of missing entity combinations. Both weight functions depend on the actual entities. Note that $w^0(\cdot)$ is required to be factorized by the dimensions. $v_{i_j}^{(j)}$ is a weight for the $(i_j)^{\text{th}}$ entity in the j^{th} dimension. This weight can

[1] Note that the algorithms do not require this constraint, arbitrary values could be used for positive and negative preferences. However the effect of using different values for these parameters is not discussed here and thus we assume zeroes and ones from this point onwards.

depend on any property of the entity. $\mu^{(j)}$ and $\gamma^{(j)}$ are constants for the j^{th} dimension. Therefore the weight by a given dimension can be either a constant or depend on a property of the actual entity. Although this sufficiently generic weight function class enables using different weighting schemes, the exploration of its effect is left to future research.

For the sake of simplicity, here I use a simple weight function by setting $\mu^{(j)} = 0$ and $\gamma^{(j)} = 1$ for all j, and setting $w^1(i_1, \dots, i_{N_D}) = \alpha \cdot \#(i_1, \dots, i_{N_D})$. That is $w^0(\cdot) = w_0 = 1$ for every entity combination and $w^1(\cdot)$ is proportional with the number of occurrences of the corresponding combination in the training set. This basic weighting assumes that entity combinations are missing at completely random [38] and that it is more important to accurately predict for entity combinations with actual feedback than for ones with no feedback. This weighting scheme is the generalization of the concept introduced in [30]:

$$\mathcal{W}(i_1, \dots, i_{N_D}) = \begin{cases} w^1(i_1, \dots, i_{N_D}) = \alpha \cdot \#(i_1, \dots, i_{N_D}) \gg w_0, & \text{if } r_{i_1,\dots,i_{N_D}} = 1 \\ w^0(i_1, \dots, i_{N_D}) = w_0, & \text{otherwise} \end{cases} \tag{4.2}$$

The algorithms assign one feature matrix to each dimension of the tensor. The feature matrix $M^{(d)}$ for the d^{th} dimension is of size $K \times S_d$. The columns of the feature matrix are the feature vectors assigned to the entities of the dimension. K is the number of features that is a parameter of the algorithms. The feature vectors are used to predict the preferences of a user on an item under the configuration of contexts. The algorithms optimize by a loss function using an optimization strategy.

Both algorithms use pointwise ranking, i.e. the estimation of the preference values in R with their importance determined by $\mathcal{W}$. The weighted sum of squared loss is appropriate for this task, hence I use it as the loss function of iTALS and iTALSx. To lessen the effects of overfitting I use ℓ_2 regularization by adding the sum of the squares of model parameters to the loss:

$$L = \sum_{i_1=1,\dots,i_{N_D}=1}^{S_1,\dots,S_{N_D}} \mathcal{W}(i_1, \dots, i_{N_D})(r_{i_1,\dots,i_{N_D}} - \hat{r}_{i_1,\dots,i_{N_D}})^2 + \sum_{d=1}^{N_D} \sum_{i=1}^{S_i} \lambda_{d,i} \left|\left|M_i^{(d)}\right|\right|_2^2 \tag{4.3}$$

Note that the regularization coefficient can be different for different feature vectors. Here I use coefficients proportional to the support of the given entity with different ratios for dimensions, similarly to [48]. Thus $\lambda_{d,i} = \lambda_d \cdot N_i^{(d)+}$ and λ_d values are hyperparameters of the algorithm.

Optimization is done via Alternating Least Squares (ALS). In ALS, feature matrices are computed in an alternating fashion. One matrix is computed at a given time and all but the currently computed matrix are fixed. The efficient usage of ALS with implicit problems is not straightforward, although it is possible with the smart separation of computations.

The key difference between iTALS and iTALSx is the preference model, i.e. the expression used to compute $\hat{r}_{i_1,\dots,i_{N_D}}$. This however also affects how the computations can be done efficiently. The following two chapters describe preference models and the derivation of the algorithms in details.

4.2 iTALS

The iTALS algorithm estimates the preferences of user u on item i under the given values of the context dimensions as sum of the values in the Hadamard products (also known as elementwise product) of the corresponding feature vectors. To be less precise, the preference is given by the dot product between the N_D corresponding vectors. This model is referred to as the N-way interaction model (or N-way model for short). The following expression describes the model formally:

$$\hat{r}_{i_1,\dots i_{N_D}} = 1^T \left(M^1_{i_1} \circ M^2_{i_2} \circ \dots \circ M^{N_D}_{i_{N_D}} \right) + \sum_{d=1}^{N_D} b^{(d)}_{i_d} \tag{4.4}$$

Note that the expression for $\hat{r}$ contains biases ($b^{(d)}_{i_d}$). Latent feature models often use biases to capture offsets in the data and thus be able to learn better. However biases can be incorporated in the feature vectors by reserving an additional coordinate for each dimension in the feature vectors (i.e. using feature vectors of $K + N_D$ length) and setting all but the d^{th} new parameter in these vectors to one in the d^{th} dimension's feature matrix. This way, the d^{th} feature of a feature vector in the feature matrix of the d^{th} dimension will contain the bias value. With careful computations, the usage of feature vectors of size $K+1$ is sufficient. Thus computing the model with K features and bias can be traced back to computing a biasless model with $K + 1$ features. Therefore, I will exclude biases from the models and computations from now on, in order to have a clearer presentation. Thus, the model of iTALS is described by the following expression (which can implicitly include the biases):

$$\hat{r}_{i_1,\dots i_{N_D}} = 1^T \left(M^1_{i_1} \circ M^2_{i_2} \circ \dots \circ M^{N_D}_{i_{N_D}} \right) \tag{4.5}$$

Figure 4.1 depicts the concept behind the model for $N_D = 3$. Generally, this model assumes that all dimensions interact with every other dimension and their interaction results in a preference value. From the recommendation perspective, this model reweights the user–item interaction with a context-configuration dependent feature vector (that is the product of more than one feature vectors if $N_D > 3$).

4.2.1 Derivation of iTALS

Following the ALS scheme, all but one matrix is fixed at a given time during the learning. All feature matrix computations follow the exact same steps. Therefore I demonstrate the steps for computing $M^{(1)}$ (i.e. the first feature matrix) out of convenience. The loss from equation (4.3) is convex in the non-fixed parameters (i.e. $M^{(1)}$), therefore it reaches its minimum in $M^{(1)}$ where its partial derivative with respect to $M^{(1)}$ is zero. The partial derivative of L in $M^{(1)}$ is separable by the columns of the feature matrix, thus each feature vector can be computed separately. Without the loss of generality, I show the steps for computing the first feature vector ($M^{(1)}_1$)) of the feature matrix. The steps for other feature vectors are the same.

The partial derivative of L with respect to $M^{(1)}_1$:

$$\frac{\partial L}{\partial M^{(1)}_1} = -2 \left(r_{1,i_2,\dots,i_{N_D}} - \hat{r}_{1,i_2,\dots,i_{N_D}} \right) \frac{\partial \hat{r}_{1,i_2,\dots,i_{N_D}}}{\partial M^{(1)}_1} + 2\lambda_{1,1} M^{(1)}_1 \tag{4.6}$$

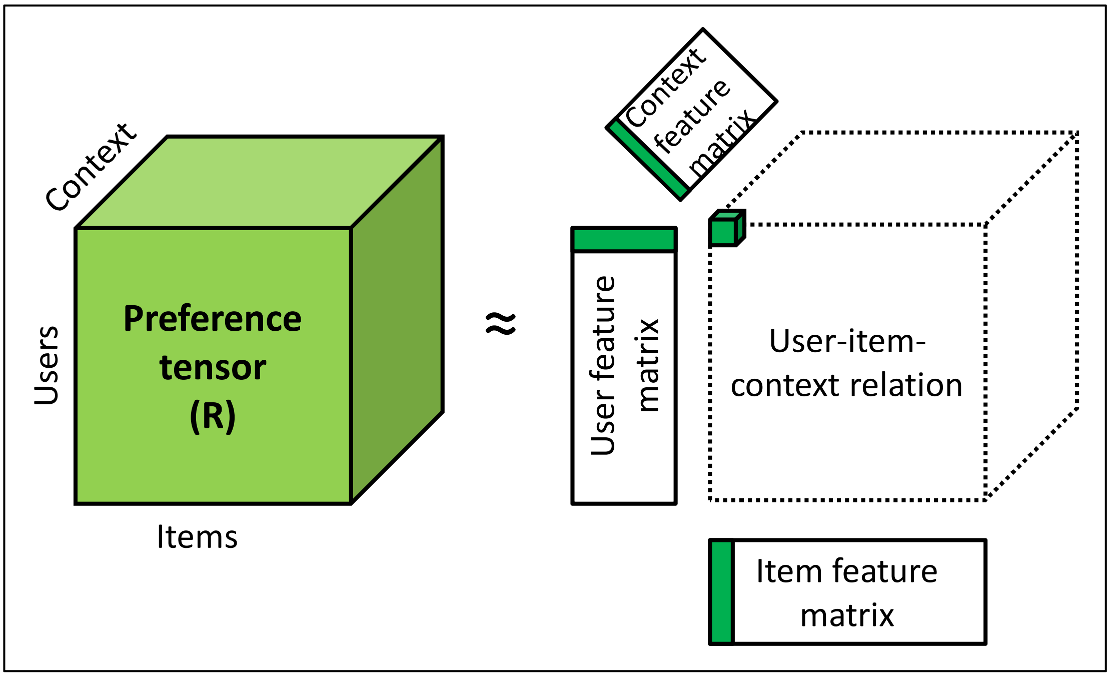

Figure 4.1: Concept of the N-way model of iTALS with 3 dimensions of the classical user–item–context setting.

By substituting the model from equation (4.5):

$$\begin{aligned}\frac{\partial L}{\partial M_1^{(1)}} = &-2\underbrace{\sum_{i_2=1,\ldots,i_{N_D}=1}^{S_2,\ldots,S_{N_D}} \mathcal{W}(1,i_2,\ldots,i_{N_D}) r_{1,i_2\ldots,i_{N_D}} \left(M_{i_2}^{(2)}\circ\cdots\circ M_{i_{N_D}}^{(N_D)}\right)+}_{\mathcal{O}} \\ &+2\underbrace{\sum_{i_2=1,\ldots,i_{N_D}=1}^{S_2,\ldots,S_{N_D}} \mathcal{W}(1,i_2,\ldots,i_{N_D}) \left(M_{i_2}^{(2)}\circ\cdots\circ M_{i_{N_D}}^{(N_D)}\right)\left(M_{i_2}^{(2)}\circ\cdots\circ M_{i_{N_D}}^{(N_D)}\right)^T M_1^{(1)}+}_{\mathcal{I}} \\ &+2\lambda_{1,1}M_1^{(1)}\end{aligned} \tag{4.7}$$

The computation of $\mathcal{O}$ (i.e. the first part of (4.7)) is efficient since most of the members in the sum are zeroes, because most of the preference values are zeroes[2]. However the computation of $\mathcal{I}$ is expensive as it requires summing over every possible entity combination from the $2\ldots N_D$ dimensions. Therefore the computation of $\mathcal{I}$ is separated into two parts by using $\mathcal{W}(i_1,i_2,\ldots,i_{N_D}) = w_0 + w'(i_1,i_2,\ldots,i_{N_D})$:

[2] This does not apply if negative preference values are set to other values. However efficient computations are still possible, using steps similar to efficiently computing $\mathcal{I}$

$$\begin{aligned}
\mathcal{I} = & \underbrace{w_0 \sum_{i_2=1,\ldots,i_{N_D}=1}^{S_2,\ldots S_{N_D}} \left(M^{(2)}_{i_2} \circ \cdots \circ M^{(N_D)}_{i_{N_D}}\right) \left(M^{(2)}_{i_2} \circ \cdots \circ M^{(N_D)}_{i_{N_D}}\right)^T +}_{\mathcal{J}} \\
& \underbrace{+ \sum_{i_2=1,\ldots,i_{N_D}=1}^{S_2,\ldots,S_{N_D}} w'(1, i_2, \ldots, i_{N_D}) \left(M^{(2)}_{i_2} \circ \cdots \circ M^{(N_D)}_{i_{N_D}}\right) \left(M^{(2)}_{i_2} \circ \cdots \circ M^{(N_D)}_{i_{N_D}}\right)^T}_{\mathcal{J}'}
\end{aligned} \tag{4.8}$$

$\mathcal{J}'$ can be efficiently computed, as most members of this sum are zeroes, because $w'(1, i_2, \ldots, i_{N_D}) = \mathcal{W}(1, i_2, \ldots, i_{N_D}) - w_0$ is zero if there is no event for the given entity combination. Note that while $\mathcal{J}'$ depends on which feature vector is being computed, $\mathcal{J}$ is the same for all columns of $M^{(1)}$ and thus can be precomputed before computing $M^{(1)}$. The efficient computation requires us to realize that:

$$\begin{aligned}
\mathcal{J}_{j,k} = & w_0 \left(\sum_{i_2=1,\ldots,i_{N_D}=1}^{S_2,\ldots,S_{N_D}} \left(M^{(2)}_{i_2} \circ \cdots \circ M^{(N_D)}_{i_{N_D}}\right) \left(M^{(2)}_{i_2} \circ \cdots \circ M^{(N_D)}_{i_{N_D}}\right)^T \right)_{j,k} = \\
= & w_0 \sum_{i_2=1,\ldots,i_{N_D}=1}^{S_2,\ldots,S_{N_D}} \left(M^{(2)}_{j,i_2} \cdot \ldots \cdot M^{(N_D)}_{j,i_{N_D}}\right) \left(M^{(2)}_{k,i_2} \cdot \ldots \cdot M^{(N_D)}_{k,i_{N_D}}\right) = \\
= & w_0 \left(\sum_{i_2=1}^{S_2} M^{(2)}_{j,i_2} M^{(2)}_{k,i_2} \right) \cdot \ldots \cdot \left(\sum_{i_{N_D}=1}^{S_{N_D}} M^{(N_D)}_{j,i_{N_D}} M^{(N_D)}_{k,i_{N_D}} \right)
\end{aligned} \tag{4.9}$$

Thus the computation of $\mathcal{J}$ can be done as follows:

$$\mathcal{J} = w_0 \underbrace{\left(\sum_{i_2=1}^{S_2} M^{(2)}_{i_2} \left(M^{(2)}_{i_2}\right)^T \right)}_{C^{(2)}} \circ \cdots \circ \underbrace{\left(\sum_{i_{N_D}=1}^{S_{N_D}} M^{(N_D)}_{i_{N_D}} \left(M^{(N_D)}_{i_{N_D}}\right)^T \right)}_{C^{(N_D)}} \tag{4.10}$$

Note that $\mathcal{J}$ is computed from the elementwise products of $K \times K$ matrices. These matrices are the autocorrelation matrices of the feature matrices of the other dimensions. This observation entails that the autocorrelation matrices should be precomputed thus the computation of $\mathcal{J}$ will be fast. Also, the autocorrelation matrix of the d^{th} dimension must be recomputed after the corresponding feature matrix is updated.

After efficiently computing $\mathcal{I} = \mathcal{J} + \mathcal{J}'$ and $\mathcal{O}$, $\frac{\partial L}{\partial M^{(1)}_1} = 0$ can be solved for $M^{(1)}_1$ as follows[3]:

$$M^{(1)}_1 = \left(\mathcal{I} + \lambda_{1,1} I\right)^{-1} \mathcal{O} \tag{4.11}$$

The pseudocode of iTALS (implicit tensor ALS) is given in Algorithm 4.2.1. The pseudocode follows the deduction above. Autocorrelation matrices are initially computed in line 3. The column independent part of equation (4.8) is created in line 7. The

[3] With ℓ_2 regularization. Here I denotes the identity matrix.

column dependent parts are added in lines 12–18 and the desired column computed in line 20. This step also adds regularization to avoid numerical instability and overfitting of the model. After each column of $M^{(i)}$ is computed, the corresponding autocorrelation matrix, $C^{(i)}$ is also recomputed (line 22).

Algorithm 4.2.1 Pseudocode of the iTALS algorithm

```
Input:   $R$: a $N_D$ dimensional $S_1 \times \dots \times S_{N_D}$ sized tensor of zeroes and ones;
         $\mathcal{W}$: a weight function as defined in equation (4.2);
         $K$: the number of features; $E$: number of epochs;
         $\{\lambda_d\}_{d=1,\dots,N_D}$: regularization coefficients per dimension;
Output:  $\{M^{(i)}\}_{i=1,\dots,N_D}$: $K \times S_i$ sized low rank matrices;
procedure iTALS($R$, $W$, $K$, $E$, $\{\lambda_d\}$)
 1: for $i = 1, \dots, N_D$ do
 2:     $M^{(i)} \leftarrow$ Random $K \times S_i$ sized matrix
 3:     $C^{(i)} \leftarrow M^{(i)}(M^{(i)})^T$
 4: end for
 5: for $e = 1, \dots, E$ do
 6:     for $i = 1, \dots, N_D$ do
 7:         $\mathcal{J} \leftarrow w_0 \cdot C^{(1)} \circ \dots \circ C^{(i-1)} \circ C^{(i+1)} \dots \circ C^{(N_D)}$
 8:         for $j = 1, \dots, S_i$ do
 9:             $\mathcal{O} \leftarrow 0$
10:             $\mathcal{J}' \leftarrow 0$
11:             $n \leftarrow 0$
12:             for all $\{r \mid r = r_{j_1,\dots,j_{i-1},j,j_{i+1},\dots,j_{N_D}}, r \neq 0\}$ do
13:                 $w \leftarrow \mathcal{W}(j_1, \dots, j_{i-1}, j, j_{i+1}, \dots, j_D) - w_0$
14:                 $v \leftarrow M^{(1)}_{j_1} \circ \dots \circ M^{(i-1)}_{j_{1-1}} \circ M^{(i+1)}_{j_{i+1}} \circ \dots \circ M^{(D)}_{j_D}$
15:                 $\mathcal{J}' \leftarrow \mathcal{J}' + wvv^T$
16:                 $\mathcal{O} \leftarrow \mathcal{O} + (w + w_0)v$
17:                 $n \leftarrow n + 1$
18:             end for
19:             $\mathcal{I} \leftarrow \mathcal{J} + \mathcal{J}'$
20:             $M^{(i)}_j \leftarrow (\mathcal{I} + \lambda_i n I)^{-1}\mathcal{O}$
21:         end for
22:         $C^{(i)} \leftarrow M^{(i)}(M^{(i)})^T$
23:     end for
24: end for
25: return $\{M^{(i)}\}_{i=1,\dots,N_D}$
end procedure
```

4.2.2 Complexity

The complexity of one epoch (i.e. computing each matrix once) is $O(N_D N^+ K^2 + \sum_{i=1}^{N_D} S_i K^3)$ (see Table 4.1 for breakdown). iTALS scales *linearly* with the number of transactions in the training set. Due to the large number of transactions and the growth rate of the set of transactions, this property is very beneficial in practice. The algorithm scales cubically with the number of features in theory. However $N_D N^+ \gg \sum_{i=1}^{N_D} S_i$ and

K is small in practice, thus the first term dominates. Therefore the algorithm scales *quadratically* with K in practice (see Section 4.4.2 for empirical results on running times).

Table 4.1: Complexity of computations for iTALS

Task	**Complexity**	**Comments**
Computations required per column (e.g.: $M_1^{(1)}$)		
Computation of $\mathcal{O}$ and $\mathcal{J}'$	$O(N_1^{(1)+}K^2)$	$N^{(1)+}$ is the number of training events, in which the first entity of the first dimension is present (i.e. support of this entity). Due to the definitions of R and $\mathcal{W}$, each of these sums will have $N^{(1)+}$ non-zero members. The computation of the update vector in line 14 takes $O(KN_D)$ time, while the updates themselves (lines 15–16) take $O(K)$ and $O(K^2)$ times for $\mathcal{O}$ and $\mathcal{J}'$ respectively. In practice $N_D \ll K$.
Solving for $M_1^{(1)}$	$O(K^3)$	This requires to solve a $K \times K$ sized system of linear equations (line 20).
Total complexity of the above for all columns of $M^{(1)}$: $O(N^+K^2 + S_1K^3)$, (N^+ is the number of transactions)		
Computations once per computing a feature matrix (e.g.: $M^{(1)}$)		
Computing $\mathcal{J}$	$O(N_DK^2)$	Assembled from $C^{(i)}$ autocorrelation matrices, using all but the one corresponding to the currently computed feature matrix (line 7). (Note: $N_D \ll N^+$)
Recomputing $C^{(1)}$	$O(S_1K^2)$	Autocorrelation matrices need to be recomputed after finishing the recomputation of the feature matrix (line 22). (Note: $S_1 \ll N^+$)
Total complexity of an epoch: $O(N_DN^+K^2 + \sum_{i=1}^{N_D} S_iK^3)$		

4.2.3 Results

iTALS is compared to the non context-aware implicit matrix factorization [30], referred to here as iALS. I used seasonality and sequentiality as the context (see Section 1.4.2 for detailed description), but only one context dimension was used at a time for iTALS. A context-aware baseline, by the name of iCA was also created to separate the effects of using a context and that of using iTALS. The iCA baseline is a prefiltering method. For each context state it trains an iALS model using only the events with the appropriate context. E.g. for the VoD we defined 6 time bands within the season, thus iCA trains 6 models. The context of the recommendation request (e.g. time of day) selects the model for the prediction. This baseline treats context-states independently. Due to the separate model training, iCA is not practical when the number of context-states is high.

Table 4.2: Recommendation accuracy of the iTALS algorithm, compared to iALS and a context-aware baseline

Dataset	**K**	**iALS**	**iCA (S)**	**iTALS (S)**	**iTALS (Q)**
Grocery	20	0.0649	0.0764 (+17.74%)	0.0990 (+52.59%)	0.1220 (+88.02%)
	40	0.0714	0.0841 (+17.85%)	0.1071 (+50.01%)	0.1339 (+87.59%)
	80	0.0861	0.1072 (+24.43%)	0.1146 (+33.04%)	0.1439 (+67.05%)
TV1	20	0.1189	0.0965 (-18.88%)	0.1167 (-1.85%)	0.1417 (+19.15%)
	40	0.1111	0.0935 (-15.81%)	0.1235 (+11.20%)	0.1515 (+36.38%)
	80	0.0926	0.0825 (-10.97%)	0.1167 (+25.99%)	0.1553 (+67.60%)
TV2	20	0.2162	0.1747 (-19.20%)	0.1734 (-19.82%)	0.2322 (+7.40%)
	40	0.2161	0.1672 (-22.60%)	0.2001 (-7.41%)	0.3103 (+43.60%)
	80	0.2145	0.1615 (-24.73%)	0.2123 (-1.02%)	0.2957 (+37.82%)
LastFM	20	0.0448	0.0523 (+16.94%)	0.0674 (+50.56%)	0.1556 (+247.57%)
	40	0.0623	0.0796 (+27.84%)	0.0888 (+42.61%)	0.1657 (+166.07%)
	80	0.0922	0.1168 (+26.66%)	0.1290 (+39.90%)	0.1864 (+102.18%)
VoD	20	0.0633	0.0703 (+10.98%)	0.0778 (+22.79%)	0.1039 (+64.07%)
	40	0.0758	0.0816 (+7.74%)	0.0909 (+19.96%)	0.1380 (+82.18%)
	80	0.0884	0.0884 (+0.04%)	0.0996 (+12.73%)	0.1723 (+94.99%)

Therefore I only trained it using seasonality.

Table 4.2 shows the recommendation accuracy of iALS, iCA and iTALS w.r.t. recall@20 on five datasets using 20, 40 and 80 features. iTALS with seasonality and sequentiality is depicted as iTALS (S) and iTALS (Q) respectively. The improvement over iALS is written in brackets.

The results indicate the the usage of context indeed increases recommendation accuracy. 8 out of 15 cases iCA is better than the non context-aware iALS and is on par once. Although iCA is worse on TV1 and TV2 than iALS, it is mostly due to iCA handling the context-states independently. With sparse datasets, the separation by context-states often results in more sparse data that is harder to learn.

The accuracy of iTALS is almost always significantly higher than that of iALS (26 of 30) or iCA (14 of 15). There are four cases in which iTALS is worse than iALS: all with seasonality on TV1 with $K = 20$ and on TV2. Note however that as the number of features increases, iTALS outperforms iALS on TV1 and the difference decreases on TV2. This is due to the N-way model that is basically a weighted dot product of the user and item features where the weight vector depends on the context state. If the number of features is low, a single latent feature is more general (i.e. it blurs more aspects of the item/user together), therefore properly reweighting the user–item relation is hard. This is especially true if the training data is very sparse, which is the case with TV2.

Sequentiality resulted in better accuracy than seasonality in all of these experiments. Using sequentiality as the context for recommendations was introduced by me in [24]. The success of sequentiality can be attributed to identifying association rule like patterns of purchasing certain items one after the other. Then iTALS adapts these in a personalized manner by modifying the base user–item interaction.

4.3 iTALSx

The iTALSx algorithm is similar to iTALS, but uses a different model for preference estimation. iTALSx is originally designed to work with three dimensions (users, items and one context). The preference of user u on item i under the given value of the context dimension is predicted as the sum of the dot products between the user and item feature vector, the user and context feature vector and the item and context feature vector. This model is referred to as the pairwise interaction model or pairwise model for short. There are two generalizations of this model for higher dimensions: (1) using every possible pairwise between interactions (full pairwise interaction model, used by e.g. [51]); (2) using only user–item, user–context and item–context interactions. However neither of these generalizations are used by iTALSx, as it works with three dimensions, but these models are discussed later with GFF in Chapter 6.

The model is given by the following expression (and see Figure 4.2):

$$\hat{r}_{u,i,c} = 1^T \left(M_u^{(U)} \circ M_i^{(I)} + M_u^{(U)} \circ M_c^{(C)} + M_i^{(I)} \circ M_c^{(C)} \right) + b_u^{(U)} + b_i^{(I)} + b_c^{(C)} \tag{4.12}$$

Similarly to iTALS, the biases ($b_u^{(U)}$, $b_i^{(I)}$ and $b_c^{(C)}$) can be incorporated into the feature vectors and thus I will use the model without biases throughout the rest of the section:

$$\hat{r}_{u,i,c} = 1^T \left(M_u^{(U)} \circ M_i^{(I)} + M_u^{(U)} \circ M_c^{(C)} + M_i^{(I)} \circ M_c^{(C)} \right) \tag{4.13}$$

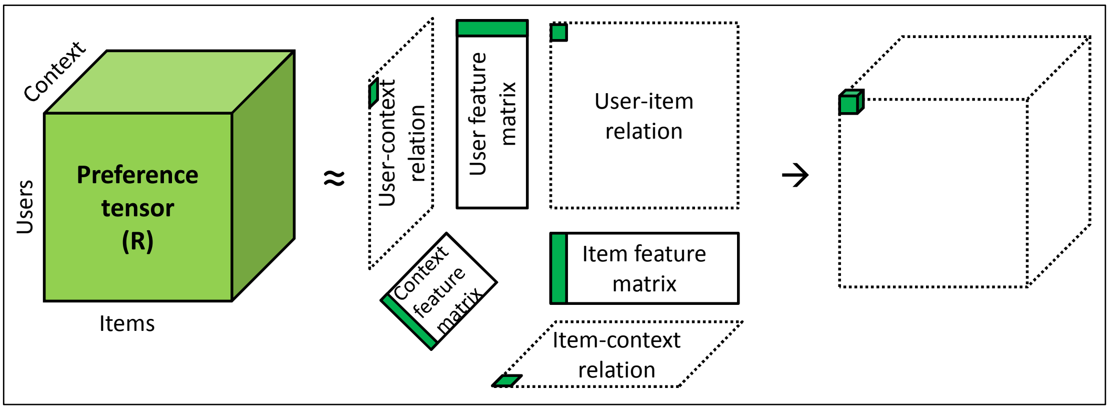

Figure 4.2: Concept of the pairwise model of iTALSx with three dimensions (user, item, context).

Note that the preference is predicted as the composite of a user–item interaction, a context dependent user bias and a context dependent item bias. The context dependent user bias (i.e. user–context interaction) does not take part in the ranking, because recommendations are generated for a given user under a given context-state, thus its value is the same for all items. However it can reduce the effect of context related shifts in the training data, which would be considered noise by a simple matrix factorization.

4.3.1 Derivation of iTALSx

The iTALSx algorithm follows the same scheme as iTALS, but the derivation differs due to using a different preference model. The model in (4.13) is symmetrical, therefore the computations for getting each feature matrix are similar. Thus only the computation of the user feature matrix $M^{(I)}$ is shown here.

The loss from equation (4.3) is convex in the non-fixed parameters (i.e. $M^{(I)}$), therefore it reaches its minimum in $M^{(I)}$ where its partial derivative with respect to $M^{(I)}$ is zero. The partial derivative of L in $M^{(I)}$ is separable by the columns of the feature matrix, thus each feature vector can be computed separately. Without the loss of generality, I show the steps for computing the first feature vector ($M_1^{(I)}$)) of the feature matrix. The steps for other feature vectors are the same.

The partial derivative of L with respect to $M_1^{(I)}$ (by substituting the model in the same step):

$$\begin{aligned}\frac{\partial L}{\partial I_i} =& -2\underbrace{\sum_{u=1,c=1}^{S_U,S_C} \mathcal{W}(u,1,c) r_{u,i,c}\left(M_u^{(U)}+M_c^{(C)}\right)}_{\mathcal{O}}+\\ &+2\underbrace{\sum_{u=1,c=1}^{S_U,S_C} \mathcal{W}(u,1,c)\left(M_u^{(U)}+M_c^{(C)}\right)\left(M_u^{(U)}+M_c^{(C)}\right)^T}_{\mathcal{I}_1} M_1^{(I)}+\\ &+2\underbrace{\sum_{u=1,c=1}^{S_U,S_C} \mathcal{W}(u,1,c)(M_c^{(C)})^T M_u^{(U)}\left(M_u^{(U)}+M_c^{(C)}\right)}_{\mathcal{I}_2}+\\ &+2\lambda_{I,1}M_1^{(I)}\end{aligned} \tag{4.14}$$

The computation of $\mathcal{O}$ is efficient since most of the members in the sum are zeroes, because most of the preference values are zeroes. The computation of $\mathcal{I}_1$ is expensive, however the same transformation can be used as with iTALS by using $\mathcal{W}(u,i,c) = w_0 + w'(u,i,c)$:

$$\begin{aligned}\mathcal{I}_1 =& w_0\underbrace{\sum_{u=1,c=1}^{S_U,S_C}\left(M_u^{(U)}+M_c^{(C)}\right)\left(M_u^{(U)}+M_c^{(C)}\right)^T}_{\mathcal{J}}+\\ &+\underbrace{\sum_{u=1,c=1}^{S_U,S_C} w'(u,1,c)\left(M_u^{(U)}+M_c^{(C)}\right)\left(M_u^{(U)}+M_c^{(C)}\right)^T}_{\mathcal{J}'}\end{aligned} \tag{4.15}$$

$\mathcal{J}'$ can be efficiently computed, as most members of this sum are zeroes, because $w'(u,1,c) = \mathcal{W}(u,1,c) - w_0$ is zero if there is no event for the given entity combination. $\mathcal{J}'$ depends on which feature vector is being computed, $\mathcal{J}$ is the same for all columns of $M^{(I)}$ and thus can be precomputed before computing $M^{(I)}$. $\mathcal{J}$ can be written in the following form:

$$\begin{aligned}\mathcal{J} =& \underbrace{w_0 \sum_{u=1}^{S_U} M_u^{(U)} \left(M_u^{(U)}\right)^T}_{C^{(U)}} + \underbrace{w_0 \sum_{c=1}^{S_C} M_c^{(C)} \left(M_c^{(C)}\right)^T}_{C^{(C)}} + \\ &+ w_0 \underbrace{\sum_{u=1}^{S_U} M_u^{(U)}}_{O^{(U)}} \underbrace{\left(\sum_{c=1}^{S_C} M_c^{(C)}\right)^T}_{\left(O^{(C)}\right)^T} + w_0 \underbrace{\sum_{c=1}^{S_C} M_c^{(C)}}_{O^{(C)}} \underbrace{\left(\sum_{u=1}^{S_U} M_u^{(U)}\right)^T}_{\left(O^{(U)}\right)^T}\end{aligned} \tag{4.16}$$

The naive computation of $\mathcal{I}_2$ is also expensive, but the same transformation can be applied as for $\mathcal{I}_1$:

$$\begin{aligned}\mathcal{I}_2 =& \underbrace{w_0 \sum_{u=1,c=1}^{S_U,S_C} (M_c^{(C)})^T M_u^{(U)} \left(M_u^{(U)} + M_c^{(C)}\right)}_{\mathcal{I}} + \\ &+ \underbrace{\sum_{u=1,c=1}^{S_U,S_C} w'(u,1,c)(M_c^{(C)})^T M_u^{(U)} \left(M_u^{(U)} + M_c^{(C)}\right)}_{\mathcal{I}'}\end{aligned} \tag{4.17}$$

$\mathcal{I}'$ can be efficiently computed similarly to $\mathcal{J}'$ and $\mathcal{I}$ can be written in the following form:

$$\begin{aligned}\mathcal{I} =& w_0 \underbrace{\sum_{u=1}^{S_U} M_u^{(U)} \left(M_u^{(U)}\right)^T}_{C^{(U)}} \underbrace{\sum_{c=1}^{S_C} M_c^{(C)}}_{O^{(C)}} + \\ &+ w_0 \underbrace{\sum_{c=1}^{S_C} M_c^{(C)} \left(M_c^{(C)}\right)^T}_{C^{(C)}} \underbrace{\sum_{u=1}^{S_U} M_u^{(U)}}_{O^{(U)}}\end{aligned} \tag{4.18}$$

Thus, both $\mathcal{J}$ and $\mathcal{I}$ can be computed from the autocorrelation matrices ($C^{(U)}$ and $C^{(C)}$) and the sum of feature vectors ($O^{(U)}$ and $O^{(C)}$) of the other two feature matrices. These statistics can be efficiently computed and the assembly from $\mathcal{J}$ and $\mathcal{I}$ is also efficient (see Section 4.3.2).

After efficiently computing $\mathcal{I}_1 = \mathcal{J} + \mathcal{J}'$, $\mathcal{I}_2 = \mathcal{I} + \mathcal{I}'$ and $\mathcal{O}$, $\frac{\partial L}{\partial M_1^{(I)}} = 0$ can be solved for $M_1^{(I)}$ as follows:

$$M_1^{(I)} = \left(\mathcal{I}_1 + \lambda_{I,1} I\right)^{-1} \left(\mathcal{O} - \mathcal{I}_2\right) \tag{4.19}$$

The pseudocode of iTALSx is given in Algorithm 4.3.1. The pseudocode follows the deduction above. Autocorrelation matrices and sum of feature vectors are initially computed in lines 3 and 4. The column independent parts $\mathcal{J}$ and $\mathcal{I}$ are created in lines 10 and 11. The column dependent parts are added in lines 15–24 and the desired column computed in line 27. This step also adds regularization to avoid numerical instability and overfitting of the model. After each column of $M^{(i)}$ is computed, the corresponding autocorrelation matrix $C^{(i)}$ and sum of features $O^{(i)}$ are also recomputed (lines 29–30).

Algorithm 4.3.1 Pseudocode of the iTALSx algorithm

```
Input:  $R$: a 3 dimensional $S_U \times S_I \times S_C$ sized tensor of zeroes and ones;
        $\mathcal{W}$: a weight function as defined in equation (4.2);
        $K$: the number of features; $E$: number of epochs;
        $\{\lambda_x\}_{x \in \{U,I,C\}}$: regularization coefficients per dimension;
Output: $\{M^{(x)}\}_{x \in \{U,I,C\}}$: $K \times S_x$ sized low rank matrices;
procedure iTALSx($R$, $W$, $K$, $E$, $\{\lambda_x\}$)
for $x \in \{U,I,C\}$ do
    $M^{(x)} \leftarrow$ Random $K \times S_x$ sized matrix
    $C^{(x)} \leftarrow M^{(x)}(M^{(x)})^T$
    $O^{(x)} \leftarrow \sum_{i_x=1}^{S_x} M_{i_x}^{(x)}$
end for
for $e = 1, \ldots, E$ do
    for $x \in \{U,I,C\}$ do
        $y \in \{U,I,C\}$, $y \neq x$ //a dimension (not the currently computed one)
        $z \in \{U,I,C\}$, $z \neq x$ and $z \neq y$ //the third dimension
        $\mathcal{J} \leftarrow w_0 \left(C^{(y)} + C^{(z)} + O^{(y)} \left(O^{(z)}\right)^T + O^{(z)} \left(O^{(y)}\right)^T\right)$
        $\mathcal{I} \leftarrow w_0 \left(C^{(y)} O^{(z)} + C^{(z)} O^{(y)}\right)$
        for $j = 1, \ldots, S_x$ do
            $\mathcal{O}, \mathcal{J}', \mathcal{I}' \leftarrow 0$
            $n \leftarrow 0$
            for all $\{r \mid$ events of the $j^{\text{th}}$ entity in the $x$ dimension$\}$ do
                $(u,i,c) \leftarrow$ indices of $r$
                $j_y, j_z \leftarrow$ indices of $r$ in the $y$ and $z$ dimensions
                $w \leftarrow \mathcal{W}(u,i,c) - w_0$
                $v \leftarrow M_{j_y}^{(y)} + M_{j_z}^{(z)}$
                $\mathcal{J}' \leftarrow \mathcal{J}' + w v v^T$
                $\mathcal{I}' \leftarrow \mathcal{I}' + w \left(M_{j_y}^{(y)}\right)^T M_{j_z}^{(z)} v$
                $\mathcal{O} \leftarrow \mathcal{O} + (w + w_0) v$
                $n \leftarrow n + 1$
            end for
            $\mathcal{I}_1 \leftarrow \mathcal{J} + \mathcal{J}'$
            $\mathcal{I}_2 \leftarrow \mathcal{I} + \mathcal{I}'$
            $M_j^{(x)} \leftarrow (\mathcal{I}_1 + \lambda_x n I)^{-1}(\mathcal{O} - \mathcal{I}_2)$
        end for
        $C^{(x)} \leftarrow M^{(x)}(M^{(x)})^T$
        $O^{(x)} \leftarrow \sum_{i_x=1}^{S_x} M_{i_x}^{(x)}$
    end for
end for
return $\{M^{(x)}\}_{x \in \{U,I,C\}}$
end procedure
```

4.3.2 Complexity

The complexity of one epoch is $O\left(N^+K^2 + (S_U + S_I + S_C)K^3\right)$ (see Table 4.3 for breakdown). iTALSx scales *linearly* with the number of transactions in the training set. Due to the large number of transactions and the growth rate of the set of transactions, this property is very beneficial in practice. The algorithm scales cubically with the number of features in theory. However $N^+ \gg (S_U+S_I+S_C)$ and K is small in practice, thus the first term dominates. Therefore the algorithm scales *quadratically* with K in practice (see Section 4.4.2 for empirical results on running times).

Table 4.3: Complexity of computations for iTALSx

Task	Complexity	Comments
Computations required per column (e.g.: $M_1^{(I)}$)		
Computation of $\mathcal{O}$, $\mathcal{J}'$ and $\mathcal{I}'$	$O(N_1^{(I)+}K^2)$	$N^{(I)+}$ is the number of training events, in which the first entity of the item dimension is present (i.e. support of this entity). Due to the definitions of R and $\mathcal{W}$, each of these sums will have $N^{(I)+}$ non-zero members. The computation of the update vector in line 19 takes $O(K)$ time, while the updates themselves (lines 20–22) take $O(K^2)$, $O(K^2)$ amd $O(K)$ times for $\mathcal{J}'$, $\mathcal{I}'$ and $\mathcal{O}$ respectively.
Solving for $M_1^{(I)}$	$O(K^3)$	This requires to solve a $K \times K$ sized system of linear equations (line 27).
Total complexity of the above for all columns of $M^{(I)}$: $O(N^+K^2 + S_IK^3)$, (N^+ is the number of transactions)		
Computations once per computing a feature matrix (e.g.: $M^{(I)}$)		
Computing $\mathcal{J}$ and $\mathcal{I}$	$O(K^2)$	Assembled from $C^{(U)}$ and $C^{(C)}$ autocorrelation matrices and $O^{(U)}$ and $O^{(C)}$ sums of feature vectors (lines 10–11).
Recomputing $C^{(I)}$ and $O^{(I)}$	$O(S_IK^2)$	Autocorrelation matrices and sums of feature vectors need to be recomputed after finishing the recomputation of the feature matrix (lines 29–30).
Total complexity of an epoch: $O\left(N^+K^2 + (S_U + S_I + S_C)K^3\right)$		

4.3.3 Results

iTALSx is compared to iALS and iCA. Only one context dimension was used at a time for iTALSx.

Table 4.4 shows the recommendation accuracy of iALS, iCA and iTALSx w.r.t. recall@20 on five datasets using 20, 40 and 80 features. iTALS with seasonality and

Table 4.4: Recommendation accuracy of the iTALSx algorithm, compared to iALS and a context-aware baseline

Dataset	K	iALS	iCA (S)	iTALSx (S)	iTALSx (Q)
Grocery	20	0.0649	0.0764 (+17.74%)	0.1027 (+58.35%)	0.1182 (+82.29%)
	40	0.0714	0.0841 (+17.85%)	0.1164 (+63.07%)	0.1299 (+81.92%)
	80	0.0861	0.1072 (+24.43%)	0.1406 (+63.23%)	0.1431 (+66.14%)
TV1	20	0.1189	0.0965 (-18.88%)	0.1248 (+4.92%)	0.1524 (+28.18%)
	40	0.1111	0.0935 (-15.81%)	0.1127 (+1.46%)	0.1417 (+27.53%)
	80	0.0926	0.0825 (-10.97%)	0.0942 (+1.67%)	0.1295 (+39.77%)
TV2	20	0.2162	0.1747 (-19.20%)	0.2220 (+2.69%)	0.2393 (+10.68%)
	40	0.2161	0.1672 (-22.60%)	0.2312 (+6.98%)	0.2866 (+32.61%)
	80	0.2145	0.1615 (-24.73%)	0.2223 (+3.62%)	0.3006 (+40.12%)
LastFM	20	0.0448	0.0523 (+16.94%)	0.0503 (+12.33%)	0.1675 (+274.35%)
	40	0.0623	0.0796 (+27.84%)	0.0599 (-3.85%)	0.1869 (+200.18%)
	80	0.0922	0.1168 (+26.66%)	0.0928 (+0.67%)	0.1984 (+115.18%)
VoD	20	0.0633	0.0703 (+10.98%)	0.0790 (+24.69%)	0.0821 (+29.67%)
	40	0.0758	0.0816 (+7.74%)	0.0916 (+20.87%)	0.1068 (+40.93%)
	80	0.0884	0.0884 (+0.04%)	0.0990 (+11.99%)	0.1342 (+51.88%)

sequentiality is depicted as iTALSx (S) and iTALSx (Q) respectively. The improvement over iALS is written in brackets.

The accuracy of iTALSx is almost always significantly higher than that of iALS (29 of 30) or iCA (12 of 15). iTALSx performs the worst (relative to iALS and iCA) on the LastFM dataset with seasonality. LastFM is a more complex dataset with relatively higher density (compared to e.g. TV2). iTALSx models the preference as the sum of two dimensional projections of a three dimensional problem. This can not capture the whole picture well and thus falls behind on more complex data.

Sequentiality proved to be more useful in increasing the accuracy than seasonality with iTALSx as well.

4.4 Comparison of iTALS and iTALSx

iTALS and iTALSx behaved differently on the five test datasets. In this section I compare iTALS and iTALSx. The full N-way model of iTALS approximates the elements in R with the sum of the elements in the Hadamard products of three vectors, while the pairwise model of iTALSx uses the sum of pairwise dot products. Mathematically the model of iTALS contains the iTALSx model as:

$$\begin{aligned}
\hat{T}^{\mathrm{iTALS}}_{u,i,c} &= 1^T \left(U_u \circ I_i \circ C_c\right) \\
3 \cdot \hat{T}^{\mathrm{iTALS}}_{u,i,c} &= 1^T \left(U_u \circ I_i \circ C_c\right) + 1^T \left(U_u \circ I_i \circ C_c\right) + 1^T \left(U_u \circ I_i \circ C_c\right) \\
\hat{T}^{\mathrm{iTALSx}}_{u,i,c} &= 1^T \left(I_i \circ C_c\right) + 1^T \left(U_u \circ C_c\right) + 1^T \left(U_u \circ I_i\right) \\
\hat{T}^{\mathrm{iTALSx}}_{u,i,c} &= 1^T \left(1 \circ I_i \circ C_c\right) + 1^T \left(U_u \circ 1 \circ C_c\right) + 1^T \left(U_u \circ I_i \circ 1\right),
\end{aligned} \tag{4.20}$$

where ∘ denotes the Hadamard product of the argument vectors.

It is more interesting to compare the models from the recommendation aspect. The main goal of a collaborative filtering algorithm is to learn the user–item relations (e.g. which user likes which item). iTALS adds context to the model and approximates the user–item relation in the 3 dimensional space. It reweights the user–item relations by a context-state dependent vector, which becomes more accurate with more factors. On the other hand, iTALSx uses a composite model and approximates the user–item relation by the sum of approximations in user–item, user–context and item–context sub-spaces, where the feature vectors in the sub-spaces are constrained by requiring a single feature vector for each entity. Consequently, the descriptive power of iTALS is larger, which can be however only leveraged at a sufficiently fine resolution of the feature space, requiring many factors. At low factor models, the boundaries of different characteristics is blurred by reweighting and the model becomes less precise. In such cases, iTALSx is expected to be more accurate, since the sub-space models can be learned easier.

4.4.1 Accuracy comparison

The significantly better between iTALS and iTALSx in the same setting (i.e.: same context, number of features, dataset) is highlighted by a light gray background in Table 4.5. Generally, iTALSx performs better if the number of features is lower. Also, there seems to be a loose connection between the density of the dataset and the relative accuracy of the two models. With a given context, iTALSx performs better if the density of the dataset is lower. High sparsity (lower density) is a common property of real life datasets, therefore iTALSx is beneficial for practical applications.

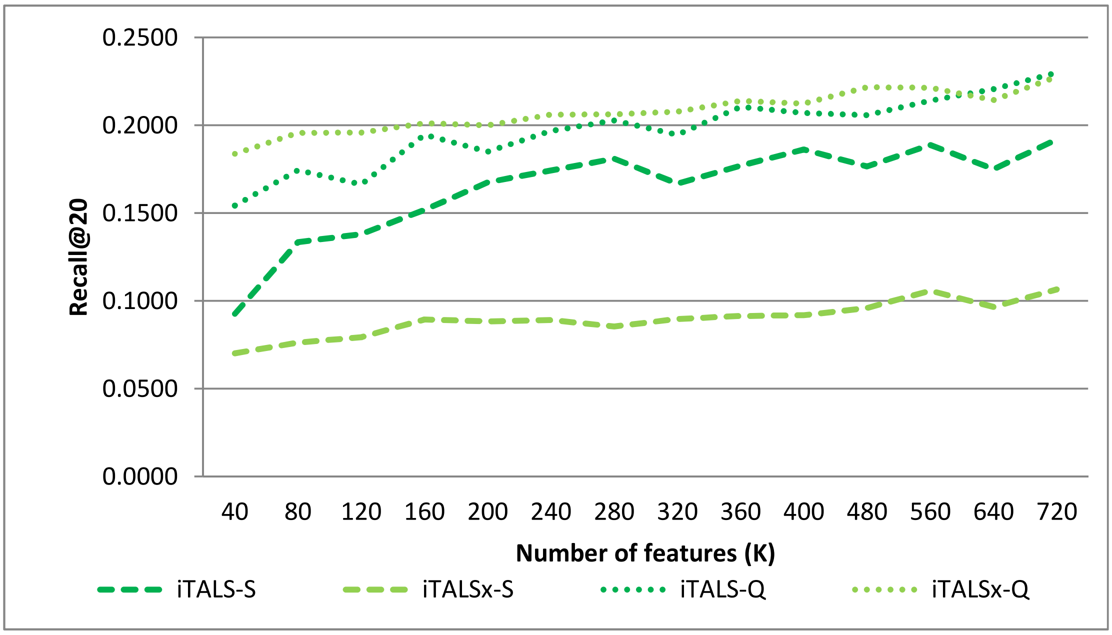

Figure 4.3: Recall@20 values for iTALS and iTALSx with seasonality (-S) and sequentiality (-Q) with the number of features ranging from 40 to 720 on the LastFM dataset.

Figure 4.3 compares iTALS and iTALSx using high number of features on the

Table 4.5: Recall@20 values for iTALS, iTALSx. Measurements with seasonality and sequentiality are denoted with the (S) and (Q) postfix respectively. The significantly better between iTALS and iTALSx in the same setting is highlighted by a light gray background

Dataset	K	iTALS (S)	iTALSx (S)	iTALS (Q)	iTALSx (Q)
Grocery	20	0.0990	0.1027	0.1220	0.1182
	40	0.1071	0.1164	0.1339	0.1299
	80	0.1146	0.1406	0.1439	0.1431
TV11	20	0.1167	0.1248	0.1417	0.1524
	40	0.1235	0.1127	0.1515	0.1417
	80	0.1167	0.0942	0.1553	0.1295
TV2	20	0.1734	0.2220	0.2322	0.2393
	40	0.2001	0.2312	0.3103	0.2866
	80	0.2123	0.2223	0.2957	0.3006
LastFM	20	0.0674	0.0503	0.1556	0.1675
	40	0.0888	0.0599	0.1657	0.1869
	80	0.1290	0.0928	0.1864	0.1984
VoD	20	0.0778	0.0790	0.1039	0.0821
	40	0.0909	0.0916	0.1380	0.1068
	80	0.0996	0.0990	0.1723	0.1342

LastFM dataset. With seasonality, iTALS is already better than iTALSx, even with 40 features. The recommendation accuracy of iTALS improves faster as the number of features increases. With sequentiality, iTALSx starts off with significantly better accuracy, but as the number of features increase, the difference becomes less significant and it disappears at high factor models. The speed of accuracy improvement is better for iTALS in both cases.

iTALSx performs better than iTALS on the sparser datasets. I used the VoD dataset (with seasonality) to further investigate the connection between sparsity and accuracy. I chose this dataset, because both methods perform similarly on it. Figure 4.4 plots recommendation accuracy (recall@20) on gradually thinned version of the dataset. At each point a certain portion of the events was randomly removed, however it was ensured that the number of active users and items do not change, thus the size of the user–item matrix is fixed. In each step the number of events is decreased by 10% of the original number until only 10% remains. I also checked the results on 5% data and on the minimal number of events where the number of active users and items remain unchanged. Note that there is a minor repetition in the VoD data and this thinning procedure also reduces repetitiveness. I checked it separately and the effect of changing the repetition in and of itself is negligible, thus changes can be attributed solely to the change in data density. I used three different number of features (5, 20 and 80).

With $K = 80$, accuracy slowly decreases for both methods as density decreases. Then at a certain point iTALS starts to lose accuracy faster and consistently underperforms

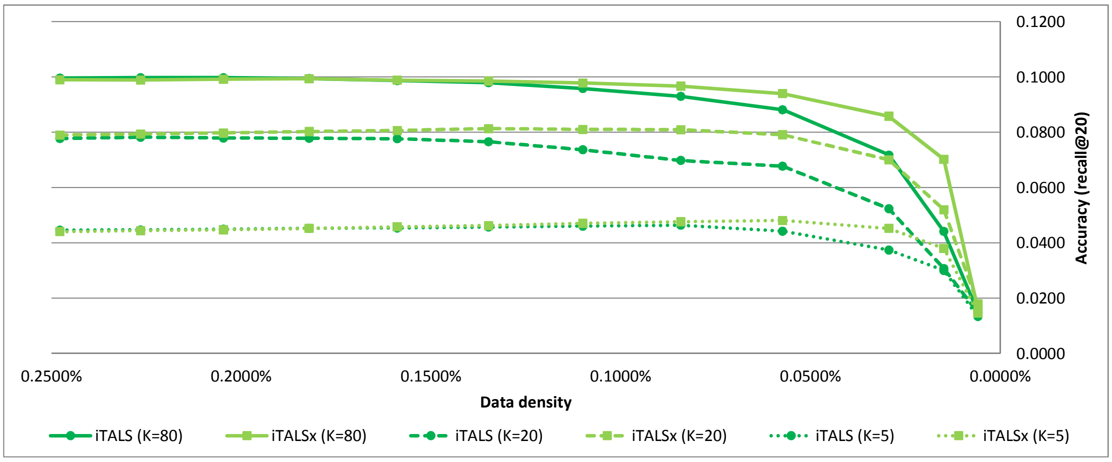

Figure 4.4: Recommendation accuracy of iTALS and iTALSx versus data sparsity. Experiments were executed on the VoD dataset by gradually removing random subsets of the events to make it sparser.

iTALSx. When the dataset is very sparse, the rate of decreasing of the accuracy becomes faster for both methods, but the effect on iTALS is larger. The lines meet once again where the minimal number of events was used to train the algorithms. It is to be expected, because the number of events per user and item is close to 1 at this point that makes collaborative filtering unviable in general. Generally, one can conclude that iTALSx performs better on sparser datasets than iTALS. The methods behave similarly with $K = 20$ and $K = 5$, however an additional interesting observations can be made.

With $K = 20$ the accuracy of iTALSx slightly increases in the first half of the graph; with $K = 5$ this is true for both methods. As data becomes more and more sparse it becomes noisier, but it also becomes less complex. The less complex the data is, the easier it is to model the user–item relations in general. This does not mean that individual users (especially those whose events were removed) can also be modelled better, but the system of events as a whole is easier to represent in the latent feature space. The representational capability of the factorization is determined by the number of features and the model. The N-way model has larger capacity as it models the entirety of the 3-way interaction; and larger number of features also means larger representational capacity. The graphs increase, when the original representational capacity is not enough to model the more complex system (i.e. more dense data). But this can not continue forever as the noisiness also increases as data becomes sparser and thus the accuracy will eventually start to decrease.

To summarize, the blurring effect of the low feature models makes learning difficult for iTALS, especially if the dataset is sparse. Sparser datasets are generally more noisy, and the elementwise model is more sensitive to noise by nature, because of the reweighting of the user–item relation in that model. My assumption about the learning capabilities of the algorithms and their connection to the finer representation of entities are underpinned as iTALS can outperform iTALSx when the number of features is sufficiently large or if the dataset is more dense. These results imply that one should use iTALSx when the dataset is sparse and we can not afford high feature models (that is most common in practical applications).

4.4.2 Complexity and training times

The complexity is $O(N^+K^2 + (S_U + S_I + S_C)K^3)$ for both iTALS and iTALSx for three dimensions. Since in practice $N^+ \gg (S_U + S_I + S_C)$, each method scales with K^2 when low-factor models are used. However the training time of iTALSx is slightly higher, because (a) iTALS does not require $O^{(X)}$ (i.e. the sum of the feature vectors of the feature matrix of the X dimension) for its computations; (b) the computations in iTALSx require a few extra operations (see Figure 4.5).

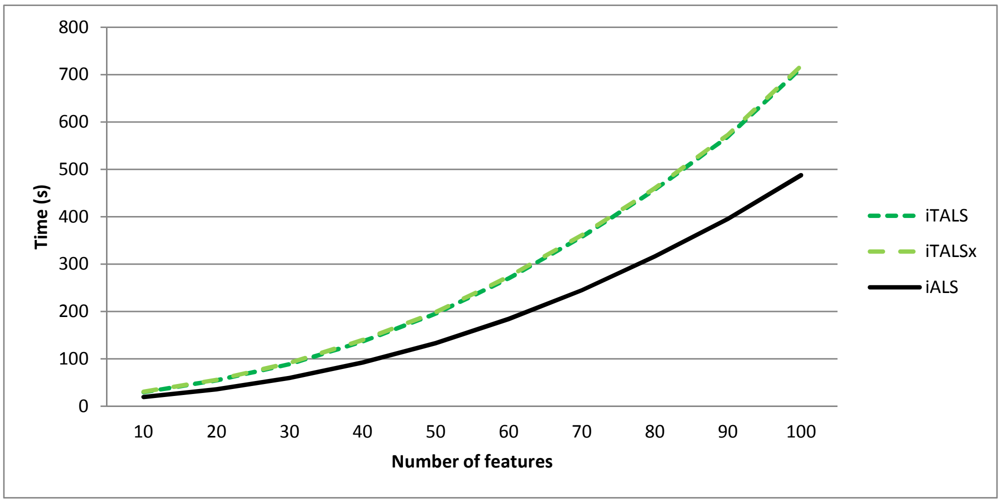

Figure 4.5: The time of one epoch (computing each feature matrix once) with iTALS and iTALSx using different number of features. (Measurements on the VoD data set; only one core used.) Results for iALS are also depicted.

Figure 4.5 also contains the training times for non-context-aware (2D) iALS algorithm, that uses a similar method for learning. The complexity of iALS is $O(N^+K^2 + (S_U + S_I)K^3)$. This means that the running times of iALS and the context-aware methods differ only in a constant multiplier, that is proportional to the number of matrices to be recomputed, but the time to compute one feature matrix is virtually the same for these algorithms.

4.5 Summary

In Chapter 4 I proposed context-aware factorization algorithms for implicit data and tested them on five datasets with two context dimensions.

In Section 4.2 I proposed the iTALS algorithm to solve the implicit feedback based context-aware recommendation task. The method and the results described in this section were published in [24] and the following theses are based on them.

> **Thesis 2.1** *I developed iTALS, a tensor factorization method that uses pointwise ranking via optimizing for weighted sum of squared errors. It estimates preferences using the N-way interaction model, i.e. the sum of elements in the elementwise product of feature vectors from each dimension. I showed that iTALS can be applied to solve the implicit feedback based context-aware recommendation problem by using ones and zeroes for positive and missing feedback respectively with higher weights for positive feedback.*
>
> **Thesis 2.2** *I showed that iTALS significantly outperforms the non context-aware implicit matrix factorization and the prefiltering based context-aware baseline with respect to recommendation accuracy, measured by recall.*
>
> **Thesis 2.3** *I demonstrated that iTALS can be trained efficiently on the implicit feedback based context-aware recommendation problem, using alternating least squares. I showed that iTALS can be efficiently used in practice as it scales linearly with the number of events and quadratically with the number of features in the range of practically useful number of feature values.*

In Section 4.3 I proposed the iTALSx algorithm to solve the implicit feedback based context-aware recommendation task. The method and the results described in this section were published in [22, 23] and the following theses are based on them.

> **Thesis 3.1** *I developed iTALSx, a tensor factorization method that uses pointwise ranking via optimizing for weighted sum of squared errors. It estimates preferences using the pairwise interaction model, i.e. the sum of dot products between feature vectors from each pair of dimensions. I showed that iTALSx can be applied to solve the implicit feedback based context-aware recommendation problem by using ones and zeroes for positive and missing feedback respectively with higher weights for positive feedback.*
>
> **Thesis 3.2** *I showed that iTALSx significantly outperforms the non context-aware implicit matrix factorization and the prefiltering based context-aware baseline with respect to recommendation accuracy, measured by recall.*
>
> **Thesis 3.3** *I demonstrated that iTALSx can be trained efficiently on the implicit feedback based context-aware recommendation problem, using alternating least squares. I showed that iTALSx can be efficiently used in practice as it scales linearly with the number of events and quadratically with the number of features in the range of practically useful number of feature values.*

In Section 4.4 I experimented with the iTALS and iTALSx algorithms, compared them and identified easily accessible contexts. The results described in this section were published in [22–24] and the following theses are based on them.

Thesis 4.1 *I proposed to use sequentiality as context for recommendations. Sequentiality is the item with which the user previously interacted, before the current one. I argued that this context information is available with every dataset where transactions can be ordered based on their time of occurrence, which is common in practice. I showed that using this information can significantly increase recommendation accuracy to using no context and even to using seasonality as the context in a wide variety of settings (dataset, algorithms, models, number of features).*

Thesis 4.2 *I compared the strengths and weaknesses of iTALS (N-way model) and iTALSx (pairwise model). I found that the N-way model is more suitable when the number of features is high and/or if the dataset is denser; and the pairwise model is better otherwise.*

Chapter 5

Speeding up ALS for context-aware factorization

In this chapter I propose ways to speed-up ALS based context-aware factorization methods such as iTALS and iTALSx [25].

5.1 Improving training times for practical usefulness

The training time of the algorithms is key aspect for practical applicability. Faster training allows to (1) capture a more recent state of the system modeled (advantageous for any system, but required for ones where the lifetime of the items is short or new items appear constantly); (2) retrain the models more frequently; (3) apply trade-off between running times and accuracy by using more features or running more epochs.

A straightforward way of speeding up training – without any modification on the base algorithm – is to distribute computations between multiple processing units (e.g. processor cores, machines). A considerable advantage of most ALS based methods is that the majority of computations are independent and therefore can be done simultaneously. With iTALS/iTALSx, the feature vectors of a dimension are computed independently, therefore the degree of parallelization can be as high as the number of entities (users, items, context-states). Since the computation for one entity is fast, the method scales well. However, ALS-based methods (including iTALS) require that at least the model (feature matrices) are stored in memory and each processing unit has access to this shared memory.[1] Otherwise a huge communication overhead arises, since the computations require random access to the feature matrices. This also implies that ALS does not work well with standard map-reduce based big data technologies [6], but requires a different solution (e.g.: multicore/multiprocessor machine, GPGPU, multi-GPU systems, cluster with shared memory, etc.).

Models and data fit in memory in most cases. With the indexing overhead required by iTALS, $\sim 45\,\mathrm{M}$ 3-tuple records can be stored in 1 GB. The models take even less space: low and high factor models ($K = 40$ and $K = 200$) would require 22.13 MB and 110.63 MB,[2] respectively. Today's normal PCs therefore can handle around 1 billion

[1] It is beneficial if the data is stored in the shared memory as well, but it can be stored on disk as well, if properly indexed.

[2] Here we assumed a relatively high density of $\sim 1\%$, $100K$ for users and $45K$ for items that is realistic

records, their high end counterparts can deal with several billions and shared memory clusters are capable of working with tens or even hundreds of billion events. Thus this approach is feasible for most of the recommendation tasks.

On the other hand, there is room for improvement beyond distributing the computations. Distributed computation does not decrease the total load on the infrastructure and the training might still take long due to some computationally expensive steps in the algorithm. Usually the CPU is the bottleneck for such algorithms, in contrast to the classic big data problem. ALS slows down significantly if the number of features (K) is high. This prevents the efficient usage of high factor models. High factor models are generally more accurate than low factor ones therefore it would be beneficial to use them.

In this chapter, I propose two approximate methods that significantly speed up ALS-learning, especially if the number of features is high, that is, the gain in speed increases as the number of features increases.

The proposed methods allow for a better trade-off between speed and recommendation accuracy. One can either train a model with the same accuracy in significantly less time, train a model with more features (and thus be more accurate) with the same training time, or anything in-between. This solution does not address the incompatibility between ALS and map-reduce based big data technology, we instead offer feasible trade-off solutions using approximate methods to reduce the computational complexity of ALS-learning.

Recall that except for the matrix inversion, ALS based algorithms scale quadratically with K. For iTALS, the DN^+K^2 term dominates the $K^3\sum_{i=1}^{D} S_i$ term in the complexity when we use low-factor models, when $DN^+ \gg \sum_{i=1}^{D} S_i$. The computation of the cubical term can still take a long time, especially with higher K values or more context dimensions. Therefore, I propose two approximate solutions instead of the naive ALS to further reduce the time complexity of the learning process of algorithms like iTALS and iTALSx. ALS-CD applies coordinate descent learning, while ALS-CG adapts the conjugate gradient descent method. Both methods were integrated into iTALS and iTALSx (and GFF from Chapter 6). The adaptations are generalizations of the techniques proposed for matrix factorization in [50] and [67]. This generalization is not exclusive to the aforementioned algorithms, other ALS based factorization algorithms can benefit from the direct application of these methods. The approximate methods are presented with iTALS, using it as an example. They can be easily applied to other methods using similar steps.

I will show that the approximate variants provide a trade-off: they can achieve lower running times (see Section 5.5.2) in exchange for somewhat higher loss function values. Note that higher loss function values are not necessarily translated to lower accuracy in recommendations when one applies other, non-error based metrics (classification or ranking metrics) for the evaluation (see Section 5.5.1).

5.2 Coordinate descent

The coordinate descent (CD) approach approximates the feature vector by computing its coordinates separately. CD approximates the least squares solution of a $b = Ax$ linear

for $\sim 45\,\mathrm{M}$ record.

system (seeking x). By fixing all but one feature and computing the remaining ones, the matrix inversion can be avoided (it is reduced to a division), thus the computation time can be greatly reduced (see Algorithm 5.2.1). The complexity of this algorithm is $O\left(N_I K(K+N_E)\right)$, where N_E is the number of rows (examples) of A. Note that while the solution provided by CD can be good enough, it does not converge to the least squares solution.

Algorithm 5.2.1 Weighted coordinate descent method

Input: A: $N_E \times K$ matrix of input examples; b: output for the examples; $x^{(0)}$: initial solution; w: vector of weights; λ: regularization coefficient; N_I: number of iterations
Output: x: approximate solution of $Ax = b$

```
procedure SOLVE-WEIGHTED-CD($A$, $b$, $x_0$, $w$, $\lambda$, $N_I$)
 1: $x \leftarrow x^{(0)}$
 2: $\hat{b} \leftarrow \sum_{j=1}^{K} A_j x_j$
 3: for $c = 1, \ldots, N_I$ do
 4:     for $k = 1, \ldots, K$ do
 5:         $\hat{b} \leftarrow \hat{b} - A_k x_k$
 6:         $x_k \leftarrow \frac{\sum_{i=1}^{N_E} w_i A_{i,k}\left(b_i - \hat{b}_i\right)}{\sum_{i=1}^{N_E} w_i A_{i,k}^2 + \lambda}$
 7:         $\hat{b} \leftarrow \hat{b} + A_k x_k$
 8:     end for
 9: end for
10: return $x$
end procedure
```

The CD approximation of ALS uses Algorithm 5.2.1 to compute each feature vector instead of solving the system of linear equations directly. The biggest difficulty to adapt CD to iTALS/iTALSx (or to any other complex models) is posed by the extremely high number of examples (N_E), which corresponds to the number of rows of A. This quantity is the product of the sizes of all but one dimension, which is $N_E = \prod_{j=1,j\neq i}^{D} S_j$, when a feature vector of the i^{th} dimension is sought.

Positive examples can be decomposed into the sum of a negative (weight: w_0; preference: $p_0 = 0$) and a modified positive example (weight: $w'(\cdot) = w(\cdot) - w_0$; preference: $p_1 = 1$). Thus line 6 can be reformulated as follows:

$$x_k \rightarrow \frac{w_0 \sum_{i=1}^{N_E} A_{i,k}(p_0 - \hat{b}_i) + \sum_{i \in \text{pos.ex.}} w'_i A_{i,k}(p_1 - \hat{b}_i) + w_0 \sum_{i \in \text{pos.ex.}} (p_1 - p_0) A_{i,k}}{\lambda + w_0 \sum_{i=1}^{N_E} A_{i,k}^2 + \sum_{i \in \text{pos.ex.}} w'_i A_{i,k}^2} \tag{5.1}$$

Summing over the positive examples is fast, because their number is small. Even though the negative examples are shared between all features of all feature vectors of a given dimension, their number is too high for the efficient usage of the coordinate descent method. However the negative examples are not directly needed. They are used to compute the following statistics that are used to calculate x_k [3]:

[3] $\hat{b}_i = \sum_{j=1,j\neq k}^{K} A_{i,j} x_j$ was substituted into the first sum of (5.1); $A_{i,k} A_{i,j} x_j$ $(j \neq k)$ is used in the nominator and $A_{i,k} A_{i,k}$ is used in the denominator.

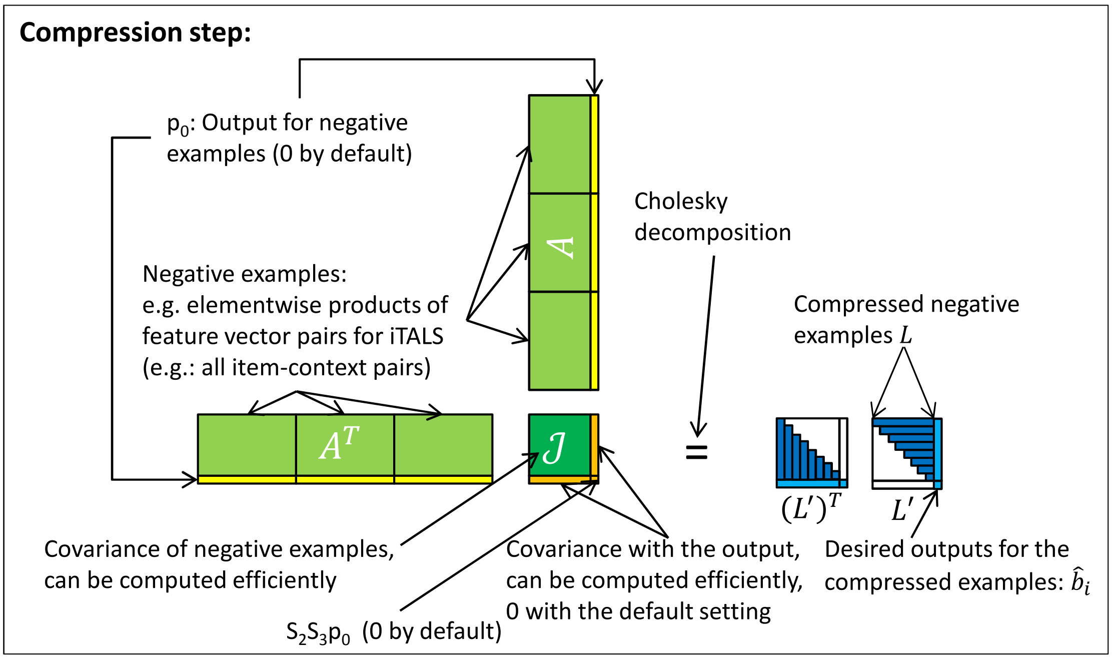
Compression step:
p_0: Output for negative examples (0 by default)
A
Cholesky decomposition
Negative examples: e.g. elementwise products of feature vector pairs for iTALS (e.g.: all item-context pairs)
Compressed negative examples L
A^T
$\mathcal{J}$
=
$(L')^T$
L'
Covariance of negative examples, can be computed efficiently
Covariance with the output, can be computed efficiently, 0 with the default setting
Desired outputs for the compressed examples: $\hat{b}_i$
$S_2S_3p_0$ (0 by default)

Figure 5.1: Concept of the compression of negative examples in the 3 dimensional user–item–context setting.

$$
\begin{aligned}
&\text{(a)}\;\; \mathrm{diag}(\mathrm{x}_1,\dots,\mathrm{x}_{\mathrm{k}-1},1,\mathrm{x}_{\mathrm{k}+1},\dots,\mathrm{x}_\mathrm{K})\mathrm{A}^\mathrm{T}\mathrm{A}_\mathrm{k} = \mathrm{D}_\mathrm{k}\mathrm{A}^\mathrm{T}\mathrm{A}_\mathrm{k} \\
&\text{(b)}\;\; p_0 \sum_{i=1}^{N_E} A_k
\end{aligned}
\tag{5.2}
$$

(a) is the k^{th} column of $\mathcal{J}$ weighted by diagonal matrix that contains the values of the actually computed feature vector where the currently computed feature set to 1. $\mathcal{J}$ can be computed efficiently using (4.10) and (4.16) for iTALS and iTALSx respectively. (b) is zero with the standard setting of preferences[4]. Using the precomputed $\mathcal{J}$ speeds up line 6 but the updating of the predicted output in lines 5 and 7 still iterates over all of the negative examples. Therefore I compress the negative examples into $K+1$ virtual examples using the following steps (also see figure 5.1):

- A vector of zeros (covariance with the output) is appended to $\mathcal{J}$ from the right and from the bottom. Thus we get a $(K+1)\times(K+1)$ sized matrix: $\hat{\mathcal{J}}$. The $(K+1, K+1)$ element of $\hat{\mathcal{J}}$ is set to $p_0 \prod_{j=1, j\neq i}^{D} S_j$, where p_0 is the value associated with the negative preference ($p_0 = 0$ by default). $\hat{\mathcal{J}}$ is symmetric and positive definite. This step is needed because the input and the output must be compressed simultaneously. (This step is the same as appending the desired output of the negative preference to the examples (i.e. a vector of p_0 values to A) and then computing the covariance of this matrix.)

[4] Even if the negative preference is considered to be not zero, this can also be precomputed in an efficient way.

- $\hat{\mathcal{J}}$ is decomposed into $(L')^T L'$ using Cholesky decomposition. L' is an upper triangular matrix. The decomposition requires $O(K^3)$ time, but has to be computed only once per recomputing a feature matrix, because $\hat{\mathcal{J}}$ is shared across all feature vectors of a dimension.

- The rows of L' are the compressed negative examples. The first K coordinates of a row are the coefficients of the compressed example and the last coordinate is the desired output for those coefficients. Due to the nature of the transformation $\mathrm{diag}(\mathrm{x}_1, \ldots, \mathrm{x}_{\mathrm{k}-1}, 1, \mathrm{x}_{\mathrm{k}+1}, \ldots, \mathrm{x}_\mathrm{K}, 1)\,(\mathrm{L}')^\mathrm{T}\,\mathrm{L}'_\mathrm{k} = (\mathrm{D}_\mathrm{k}\mathrm{A}^\mathrm{T}\mathrm{A}_\mathrm{k}|\mathrm{p}_0 \sum_{\mathrm{i}=1}^{\mathrm{N}_\mathrm{E}} \mathrm{A}_\mathrm{k})$, i.e. (a) with (b) appended to it from (5.2). Using the compressed examples the computation of (5.2) and also the updates of lines 5 and 7 can be done efficiently.

The examples of the negative feedback were compressed into $K+1$ virtual examples, that is shared for every feature vector of the i^{th} matrix. For the j^{th} feature vector the positive feedback on the j^{th} entity is also needed. The number of positive examples equals to the number of events with on the given entity. Therefore the number of examples for an entity is $K + 1 + N_j^{(i)+}$, which is low, thus the coordinate descent method can be computed efficiently.

Algorithm 5.2.2 shows the pseudocode for iTALS-CD. It is identical with algorithm 4.2.1 until line 7. In lines 8–14 the negative examples are compressed. This starts by appending a column and a row of zeros to $\mathcal{J}$. Note that zeros are used because the preference value associated to the missing negative feedback is zero. A weight vector is also needed because the algorithm optimizes for wRMSE (line 19). Updating steps of this matrix and vectors with positive examples are executed in lines 20–27. The solution for $M_j^{(i)}$ is computed in line 28 using a weighted coordinate descent method (see Algorithm 5.2.1). The signature of the solver is Solve-weighted-CD(A, b, x_0, w, λ, N_I), where the linear system is $A^T x = b$, x_0 is an initial solution, the error is weighted by weight vector w as $(\left|\left|w \circ \left(b - A^T x\right)\right|\right|)$, λ is the regularization parameter and N_I is the number of iterations.

5.2.1 Complexity

The complexity of one epoch (i.e. computing each matrix once) is $O(N_D K^3 + N_D N^+ N_I K + \sum_{i=1}^{N_D} S_i K^2)$ (see Table 5.1 for breakdown).

Comparing this to the complexity of iTALS $(O(N_D N^+ K^2 + K^3 \sum_{i=1}^{N_D} S_i))$, we can observe that iTALS-CD also scales cubically in K, however, the coefficient is reduced from $\sum_{i=1}^{D} S_i$ to N_D. The other two terms are similar, however there is $N_I K^2$ and $N_I K$ in the place of K^3 and K^2. In practice, when $N_I \ll K$ and N_D is low, for practical K values it scales *linearly* in K because $N_D N^+ \gg \sum_{i=1}^{N_D} S_i$.

5.3 Conjugate gradient

The conjugate gradient (CG) [21] method is the state-of-the-art iterative method for solving $Ax = b$ type systems of linear equations, where A is symmetric positive definite (see Algorithm 5.3.1). The geometric interpretation of CG is that first a direction is selected in which the error can be reduced the most. In the following iterations the algorithm selects the best direction that is pairwise conjugate to every previous direction.

Algorithm 5.2.2 iTALS using coordinate descent for speedup

```
Input:  $R$: a $N_D$ dimensional $S_1 \times \cdots \times S_{N_D}$ sized tensor of zeroes and ones;
        $\mathcal{W}$: a weight function as defined in equation (4.2);
        $K$: the number of features; $E$: number of epochs;
        $\{\lambda_d\}_{d=1,\ldots,N_D}$: regularization coefficients per dimension;
        $N_I$: number of inner iterations for CD;
Output: $\{M^{(i)}\}_{i=1,\ldots,N_D}$: $K \times S_i$ sized low rank matrices;
procedure iTALS-CD($T$, $W$, $K$, $E$, $\lambda$, $N_I$)
for $i = 1, \ldots, N_D$ do
    $M^{(i)} \leftarrow$ Random $K \times S_i$ sized matrix
    $C^{(i)} \leftarrow M^{(i)}(M^{(i)})^T$
end for
for $e = 1, \ldots, E$ do
    for $i = 1, \ldots, N_D$ do
        $\mathcal{J} \leftarrow w_0 \cdot C^{(1)} \circ \cdots \circ C^{(i-1)} \circ C^{(i+1)} \cdots \circ C^{(N_D)}$
        $\hat{\mathcal{J}} \in \mathbb{R}^{K+1 \times K+1}$
        $\hat{\mathcal{J}}_{1:K,1:K} \leftarrow \mathcal{J}$
        $\hat{\mathcal{J}}_{1:K+1,K+1} \leftarrow 0$
        $\hat{\mathcal{J}}_{K+1,1:K+1} \leftarrow 0$
        $(L')^T L' \leftarrow$ CHOLESKY-DECOMPOSITION($\hat{\mathcal{J}}$)
        $L \leftarrow$ strip the last column of $L'$
        $b \leftarrow$ the last column of $L'$ transposed
        for $j = 1, \ldots, S_i$ do
            $n \leftarrow 0$
            $L^{(j)} \leftarrow L$
            $b^{(j)} \leftarrow b$
            $w^{(j)} \leftarrow$ vector of $w_0$ values; same length as $b^{(j)}$
            for all $\{r \mid r = r_{j_1,\ldots,j_{i-1},j,j_{i+1},\ldots,j_{N_D}}, r \neq 0\}$ do
                $w \leftarrow \mathcal{W}(j_1, \ldots, j_{i-1}, j, j_{i+1}, \ldots, j_D) - w_0$
                $v \leftarrow M^{(1)}_{j_1} \circ \cdots \circ M^{(i-1)}_{j_{1-1}} \circ M^{(i+1)}_{j_{i+1}} \circ \cdots \circ M^{(D)}_{j_D}$
                $L^{(j)} \leftarrow$ append $v^T$ to $L^{(j)}$ from below
                $b^{(j)} \leftarrow$ append 1 to $b^{(j)}$
                $w^{(j)} \leftarrow$ append $w$ to $w^{(j)}$
                $n \leftarrow n + 1$
            end for
            $M^{(i)}_j \leftarrow$ SOLVE-WEIGHTED-CD($L^{(j)}$, $b^{(j)}$, $M^{(i)}_j$, $w^{(j)}$, $\lambda_d \cdot n$, $N_I$)
        end for
        $C^{(i)} \leftarrow M^{(i)}(M^{(i)})^T$
    end for
end for
return $\{M^{(i)}\}_{i=1,\ldots,N_D}$
end procedure
```

iTALS-CG approximates the feature vectors by replacing $M^{(i)}_j = (\mathcal{I} + \lambda_i n I)^{-1}\mathcal{O}$ in line 20 of algorithm 4.2.1 with SOLVECG(A, b, x_0, M) with $A = \mathcal{I} + \lambda_i n I$, $b = \mathcal{O}$, x_0 with

Table 5.1: Complexity of computations for iTALS using ALS-CD

Task	**Complexity**	**Comments**
Computations required per column (e.g.: $M_1^{(1)}$)		
Extending L, b and w	$O(KN_1^{(1)+})$	$N_1^{(1)+}$ is the number of training events, in which the first entity of the first dimension is present (i.e. support of this entity). For each event vector v is computed and v is appended to L. b and w are extended with a scalar value.
Solving for $M_1^{(1)}$ with CD	$O\left((K^2 + N_1^{(1)+}K)N_I\right)$	The complexity of algorithm 5.2.1 is $O(N_E N_I K)$ where N_E ins the number of examples. Here $N_E = K + N_1^{(1)+}$ as we have $N_1^{(1)+}$ positive examples and the negative ones are compressed to K size.
Total complexity of the above for all columns of $M^{(1)}$: $O(N^+ K N_I + S_1 K^2)$, (N^+ is the number of transactions)		
Computations once per computing a feature matrix (e.g.: $M^{(1)}$)		
Computing $\mathcal{J}$	$O(N_D K^2)$	Assembled from $C^{(i)}$ autocorrelation matrices, using all but the one corresponding to the currently computed feature matrix.
Cholesky decomposition of $\hat{\mathcal{J}}$	$O(K^3)$	Required once per feature matrix
Recomputing $C^{(1)}$	$O(S_1 K^2)$	Autocorrelation matrices need to be recomputed after finishing the recomputation of the feature matrix.
Total complexity of an epoch: $O(N_D K^3 + N_D N^+ N_I K + \sum_{i=1}^{N_D} S_i K^2)$		

the previous value of the feature vector and $M = \operatorname{diag}(\mathcal{I} + \lambda_i n I)$. The pseudocode of the conjugate gradient method is presented in Algorithm 5.3.1. The conjugate gradient method converges to the exact solution in at most K steps. If $N_I = K$, it provides the exact solution, however it is often sufficient to run fewer inner iterations for a good solution.

5.4 Complexity

The bottleneck of the CG method is the matrix-vector multiplication with A and the inversion of M in each iteration (see Algorithm 5.3.1).

Algorithm 5.3.1 Conjugate gradient method

Input: A: $K \times K$ symmetric positive definite matrix; b: output vector; $x^{(0)}$: initial solution; M: preconditioning matrix (e.g.: diag(A)); N_I: number of iterations
Output: x: approximate solution of $Ax = b$

```
procedure SOLVECG($A$, $b$, $x_0$, $M$, $N_I$)
$r^{(0)} \leftarrow b - Ax^{(0)}$
$z^{(0)} \leftarrow M^{-1}r^{(0)}$
$p^{(0)} \leftarrow z^{(0)}$
for $i = 0, \ldots, N_I - 1$ do
    $\alpha^{(i)} \leftarrow \frac{(r^{(i)})^T z^{(i)}}{(p^{(i)})^T A p^{(i)}}$
    $x^{(i+1)} \leftarrow x^{(i)} + \alpha^{(i)} p^{(i)}$
    $r^{(i+1)} \leftarrow r^{(i)} - \alpha^{(i)} A p^{(i)}$
    $z^{(i+1)} \leftarrow M^{-1} r^{(i+1)}$
    $\beta^{(i)} \leftarrow \frac{(z^{(i+1)})^T r^{(i+1)}}{(z^{(i)})^T r^{(i)}}$
    $p^{(i+1)} \leftarrow z^{(i+1)} + \beta_i p_i$
end for
return $x^{(N_I)}$
end procedure
```

Here $A = C^{(i,j)} + \lambda I = \mathcal{J} + \sum_{N_j^{(i)+}}^{k=1} w_k v_k v_k^T + \lambda_i n I$ and I use the Jacobi preconditioner ($M = \text{diag}(A) = \text{diag}\left(\mathcal{I} + \lambda_i n I\right)$). With careful implementation the matrix-vector multiplication takes $O(K^2 + N_j^{(i)+}K)$ time and the inversion of the diagonal M matrix takes $O(K)$ time.

It takes $O(N_I N_j^{(i)+} K + N_I K^2)$ time to compute a feature vector. This sums up to $O(N_I N^+ K + S_i N_I K^2)$ for recomputing one matrix instead of $O(S_i K^3)$, the complexity of the exact method. The total complexity of iTALS-CG is $O(N_D N^+ N_I K + N_I K^2 \sum_{i=1}^{N_D} S_i)$. Note that for iTALS-CG only $\mathcal{J}$ is needed, but $\mathcal{I}$ is not, i.e. line 15 can be omitted from algorithm 4.2.1 when using the CG solver. Thus the term $N_D N^+ K^2$ can be omitted from the computation time as well. If $N_I \ll K$ iTALS-CG scales quadratically in the number of features (instead of cubically) in theory. In practice ($N_D N^+ \ll \sum_{i=1}^{N_D} S_i$) it scales *linearly* (instead of quadratically) with the number of features for small K values. However, if $N_I \approx K$, then its complexity is the same as of the exact iTALS. Since there are differences in the constant multipliers, iTALS-CG in fact can be slower than the exact iTALS in this case.

5.5 Comparison of learning methods

In this section I compare ALS, ALS-CG and ALS-CD w.r.t. recommendation accuracy, training times and convergence. I also determine the number of inner iterations based on trade-offs between running time and accuracy. I use three algorithms — iTALS, iTALSx and iALS [30] — with all three learning methods. As mentioned before, the generalization of CG and CD learning for complex D dimensional models makes it possible to use these speed-up techniques for every ALS-based factorization.[5]

5.5.1 Recommendation accuracy

Tables 5.2, 5.3 and 5.4 show recommendation accuracy in terms of recall@20 for three algorithms (iTALS, iTALSx and iALS) with seasonality and sequentiality using various number of features. The number of inner iterations was set to 2 for both CG and CD. The approximate versions used the same regularization coefficient as the corresponding base method. It is possible that there exists a better configuration of hyperparameters for the approximate versions, but keeping the parameters the same enables fair comparison, and I found that these configurations fit also quite well for the approximate learning methods.

Some values are missing from the table, because training with CD failed sometimes. One can observe that CD is somewhat unstable if there are n-way ($n > 2$) interactions in the preference model, the size of any of the interacting dimensions is low and the number of features is high. Additional experiments with different preference models confirmed this disadvantage.

The recommendation accuracy of ALS and the approximate methods are usually very similar. There are some exceptions with moderate differences. Although the value of the loss function (wRMSE) is correlated with the evaluation metric (recall), there is no direct connection between them. Thus the approximate methods can outperform

[5] The actual speed-up and improvement in scalability depend on the efficiency of certain key steps (e.g. matrix-vector multiplication for CG). These may differ from algorithm to algorithm.

Dataset	**K**	**Seasonality**			**Sequentiality**		
		ALS	**ALS-CG**	**ALS-CD**	**ALS**	**ALS-CG**	**ALS-CD**
Grocery	**40**	0.1071	0.1065	0.1043	0.1339	0.1304	0.1317
	80	0.1146	0.1193	N/A	0.1439	0.1381	0.1426
	200	0.1312	0.1342	N/A	0.1570	0.1485	0.1540
TV1	**40**	0.1235	0.1194	N/A	0.1515	0.1521	0.1518
	80	0.1167	0.1147	N/A	0.1553	0.1511	0.1483
	200	0.1055	0.1063	N/A	0.1517	0.1520	0.1505
TV2	**40**	0.2001	0.2004	0.1972	0.3103	0.3066	0.3094
	80	0.2123	0.2102	N/A	0.2957	0.2974	0.2961
	200	0.2184	0.2111	N/A	0.2821	0.2848	0.2847
LastFM	**40**	0.0888	0.1040	0.0909	0.1657	0.1605	0.1579
	80	0.1290	0.1417	N/A	0.1864	0.1796	0.1780
	200	0.1382	0.1970	N/A	0.1784	0.2044	0.2045
VoD	**40**	0.0909	0.0913	0.0910	0.1380	0.1372	0.1347
	80	0.0996	0.1002	0.0990	0.1723	0.1720	0.1627
	200	0.1026	0.1036	0.1023	0.2116	0.2111	0.2092

(a) Results with iTALS

Table 5.2: Recommendation accuracy with iTALS.

Dataset	K	Seasonality			Sequentiality		
		ALS	ALS-CG	ALS-CD	ALS	ALS-CG	ALS-CD
Grocery	**40**	0.1164	0.1208	0.1135	0.1299	0.1272	0.1283
	80	0.1406	0.1445	0.1340	0.1431	0.1385	0.1411
	200	0.1927	0.1915	0.1842	0.1655	0.1531	0.1610
TV1	**40**	0.1127	0.1077	0.1043	0.1417	0.1410	0.1414
	80	0.0942	0.0858	0.0905	0.1295	0.1309	0.1295
	200	0.0696	0.0650	0.0688	0.1106	0.1098	0.1104
TV2	**40**	0.2312	0.2274	0.2195	0.2866	0.2846	0.2856
	80	0.2223	0.2130	0.2117	0.3006	0.3017	0.2986
	200	0.1791	0.1741	0.1807	0.3023	0.3067	0.3079
LastFM	**40**	0.0599	0.0691	0.0507	0.1869	0.1830	0.1859
	80	0.0928	0.0773	0.0708	0.1984	0.1966	0.1929
	200	0.1264	0.0907	0.0922	0.2003	0.2007	0.2006
VoD	**40**	0.0916	0.0931	0.0927	0.1068	0.1073	0.1068
	80	0.0990	0.0999	0.0986	0.1342	0.1345	0.1347
	200	0.0977	0.0980	0.0970	0.1726	0.1732	0.1728

(a) Results with iTALSx

Table 5.3: Recommendation accuracy with iTALSx.

Dataset	K	ALS	ALS-CG	ALS-CD
Grocery	**40**	0.0714	0.0745	0.0814
	80	0.0861	0.0919	0.0966
	200	0.1281	0.1298	0.1237
TV1	**40**	0.1111	0.1072	0.1074
	80	0.0926	0.0899	0.0937
	200	0.0769	0.0712	0.0799
TV2	**40**	0.2161	0.2043	0.2162
	80	0.2145	0.1906	0.2140
	200	0.1958	0.1702	0.1894
LastFM	**40**	0.0623	0.0545	0.0467
	80	0.0922	0.0902	0.0574
	200	0.1199	0.1204	0.0453
VoD	**40**	0.0758	0.0779	0.0758
	80	0.0884	0.0889	0.0878
	200	0.0928	0.0921	0.0918

(a) Results with iALS

Table 5.4: Recommendation accuracy with iALS.

Method	**Performs similarly**	**Underperforms**	**Outperforms**	**Fails**	***Total***
CG	58 (77.33%)	11 (14.67%)	6 (8%)	0 (0%)	*75 (100%)*
CD	54 (72%)	9 (12%)	3 (4%)	9 (12%)	*75 (100%)*

Table 5.5: Overview of the relation of approximate methods to ALS. Similar performance means that the difference in recall@20 is lower than 5%.

Method	**Insignificant difference**	**Worse than ALS**	**Better than ALS**	**Fails**	***Total***
CG	29 (38.67%)	28 (37.33%)	18 (24%)	0 (0%)	*75 (100%)*
CD	32 (42.67%)	26 (34.67%)	8 (10.67%)	9 (12%)	*75 (100%)*

Table 5.6: Summary of statistical significance tests comparing CG and CD to ALS. ($p = 0.05$)

the exact ALS. There are only a few examples where the difference in the accuracy is considerable, but there is no clear trend on the characteristics of these examples.

For overview on the relation of the approximate methods to ALS see Table 5.5. Since small differences in recall usually do not increase practical accuracy, a threshold of 5% was set. A method is considerably better than an other if its recall is by at least 5% larger. CG performs slightly better than CD w.r.t. recommendation accuracy similarity to the exact method (58 and 54). CG also has more (considerably) outperforming results compared to the exact ALS than CD (6 and 3). The number of underperforming cases is roughly the same for CG and CD (11 and 9). If there is any appreciable difference vs. ALS, CD usually exhibits lower performance (9 of 12), while CG outperforms ALS in more than one third of the cases (6 of 17).

Besides examining if differences between methods are considerable, I also checked if the differences are statistically significant. The test set was split into 10 parts randomly. The number of relevant and recommended items (i.e. the nominator of recall) was measured for all parts.[6] Then paired t-test was used to compare the methods with $p = 0.05$. Table 5.6 contains the aggregated results. Statistical significance based differentiating is more permissive than setting a threshold of 5%. The trends are similar as in Table 5.5: CD underperforms ALS in slightly less cases than CG does; CG has more outperforms than CD; and CD fails in 9 instances.

I also combined both comparison methodologies; Table 5.7 depicts the results. The table is very similar to Table 5.5, there are only a few considerably different cases that are not significantly different.

I can thus summarize that from the recommendation accuracy point of view, both approximate learning methods are on par with the original ALS. However, due to its stability and somewhat better accuracy, CG is the more appropriate choice than CD in general. Let us investigate how much one can gain on the training time with the approximate methods, since this can justify the use of an approximate method.

[6] With fixed list length and test set these values are proportional to the recall@20 value.

Method	Performs similarly	Underperforms	Outperforms	Fails	*Total*
CG	62 (82.67%)	10 (13.33%)	3 (4%)	0 (0%)	*75 (100%)*
CD	57 (76%)	7 (9.33%)	2 (2.67%)	9 (12%)	*75 (100%)*

Table 5.7: Comparing CG and CD to ALS when differences should be both considerably and statistically significantly better.

5.5.2 Running time

Figure 5.2 depicts the time required to run one epoch (i.e. computing each feature matrix once) of iTALS with ALS, CG and CD with different number of features using the VoD dataset. The number of inner iterations was set to 2 for CG and CD. The results provide underpinning for my earlier statements about the practical scaling of the methods in the number of features. It is clear that both approximate methods scale better than the exact ALS. The speed-up factor is ~ 10.6 for CG and ~ 2.9 for CD if $K = 200$ (and it becomes even greater for larger K values). For the more commonly used $K = 80$, the speed-up is ~ 3.5 and ~ 1.3 for CG and CD, respectively. As expected, with few features (e.g. $K = 20$), due to the computational overhead the approximate methods can be somewhat slower than the exact ALS. Summarizing: for low factor models, ALS can be an appropriate learner, while for higher factors, ALS-CG and ALS-CD offer considerable speed-up.

For high factor models, CG is significantly faster than CD. CD starts scaling superlinear much earlier (around $K = 100$ in this example) than CG (still linear for $K = 200$) and also starts off with a steeper scaling graph. The speed-up from CD to CG for $K = 80$ and $K = 200$ is ~ 2.6 and ~ 3.8, respectively.

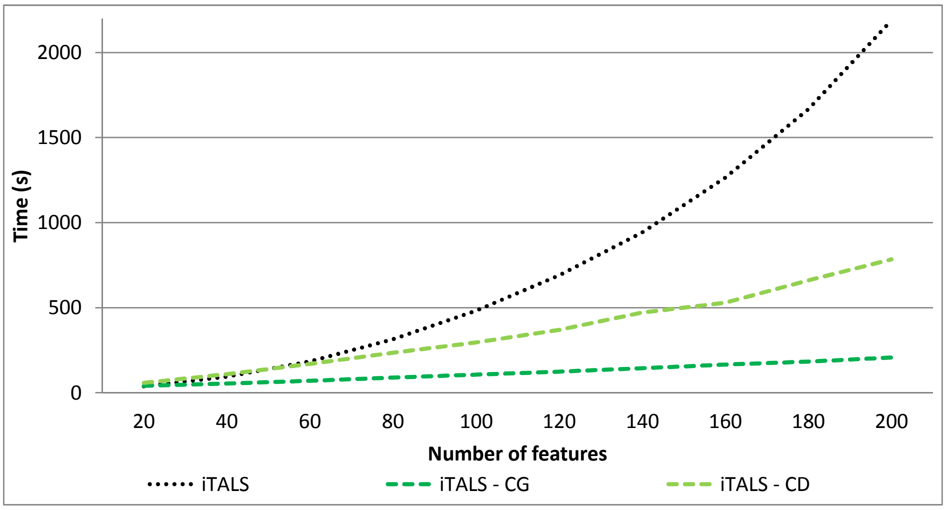

Figure 5.2: Running times of one epoch of different learning methods (ALS, CD, CG) with iTALS w.r.t. different number of feature (K) values, using one CPU core

5.5.3 Accuracy-running time trade-off

CG and CD allows for finding better trade-offs between running time and accuracy, because the (a) difference between their accuracies is usually negligible, (b) they scale better and can be trained faster than ALS and (c) models with higher number of features generally perform better. To reinforce this claim about finding better trade-offs I plotted the accuracy of iTALS versus its training time on the VoD dataset (using sequentiality as the context). The experiments were executed on a multicore processor, using 4 threads. For CG I did measurements with $K = \{20, 40, 60, 80, 120, 160, 200, 240, 320, 400, 480, 560, 640, 720\}$, for CD and ALS I stopped the at $K = 240$ and $K = 200$ respectively.

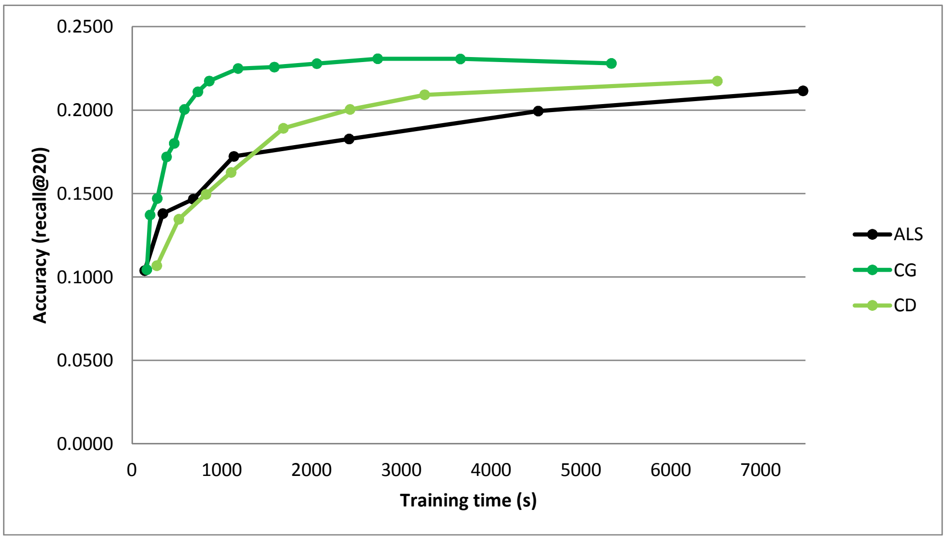

Figure 5.3: Trade-offs between accuracy and training time of iTALS, using ALS, CG and CD learning on the VoD dataset with sequentiality as the context. Executed using 4 threads with a multicore processor.

Figure 5.3 shows the results. The higher a line is on the graph, the better trade-offs are available with the corresponding method. CG is clearly the best of all three methods. It can be trained in much less time than the others and thus allows the usage of high factor models that are more accurate. Although the rate of accuracy improvement with the increasing number of features (and thus training time) slows down eventually, there is still improvement for very high factor (200+) models. In this setting, CG can train a model with $K = 320$ in the same time that is required for CD and ALS for a model with 80 factors. CD allows for slightly worse trade-offs than ALS until $K = 80$ (or training time < 1100s). This is because CD has a relatively large overhead on the computations. Note however that it is only slightly worse than ALS on this first half of the graph and due to its better scaling it offers much better trade-offs than ALS when the number of features is higher. Thus CD also generally offers better trade-offs than ALS, but it is consistently worse than CG in that regard.

Note that training time is also heavily influenced by the number of epochs. Here I ran every experiment for 10 epochs, because it is generally a good choice as these methods tend to converge in 10 epochs under a variety of different setups. I found however that the convergence of ALS, CG and CD is very similar (see below in Section 5.5.5) thus it does not affect their relations w.r.t. which method offers better trade-offs.

5.5.4 Number of inner iterations

The number of inner iterations is an important parameter of the approximate methods. Generally, the larger this value is, the more accurate the algorithms are at the cost of the increased training times. In this section I determine a good choice of this value by analyzing the trade-off between training time and accuracy.

Figure 5.4 compares the accuracy of CG and CD to ALS with different number of inner iterations. I selected two examples: one where CG and CD approximates ALS well in Tables 5.2, 5.3 and 5.4 (iTALSx, LastFM, sequentiality, $K = 80$); and one where they don't (iTALSx, LastFM, seasonality, $K = 80$). Note that the former case is more common than the latter. The behavior depicted on Figure 5.4 is general, i.e. the graphs are similar in other settings as well (not shown).

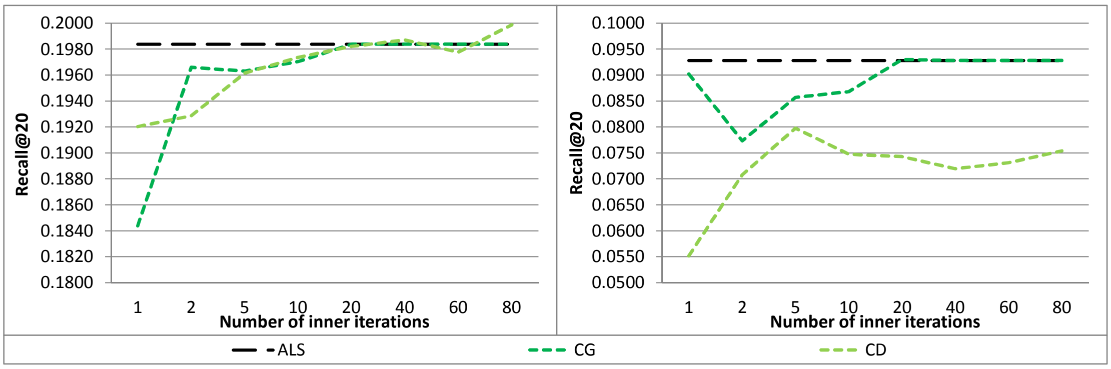

Figure 5.4: Recommendation accuracy with different number of inner iterations for CG and CD learning; value for LS solver is shown for comparison

In the first example, both approximate methods start from a lower value at $N_I = 1$. From there, their accuracy is gradually increased and CG improves slightly faster. CG and ALS compute exactly the same features if $N_I = K$. Thus CG converges to ALS as the number of inner iterations increases. They also yield the same accuracy if K is sufficiently high (20 in this case). On the other hand, CD does not converge to ALS, but gives quite similar accuracy values. It reaches the accuracy of ALS around the same $N_I = 20$ as well, and one can observe very slight variance of accuracy for $N_I > 20$. At $N_I = 80$ the accuracy becomes even slightly better than that of ALS. This is not a general behavior of CD, but as it does not converge to the exact ALS, sometimes it can give slightly better results. Note that CG can also outperform ALS (as shown in Tables 5.2, 5.3 and 5.4), but only by low N_I values. Even is CG starts off with a higher accuracy than ALS, by the increasing N_I it converges to ALS. CD can, however, theoretically outperform ALS at any N_I values.

In the second example, there is a larger difference between accuracy values of the approximate and the exact learning. CG has a relatively high accuracy at $N_I = 1$, but this is not a general behavior by any means. From $N_I = 2$, the accuracy of CG starts increasing monotonously and reaches that of the ALS around $N_I = 20$. On the other hand, the accuracy of CD varies throughout and it never even approaches that of the ALS.

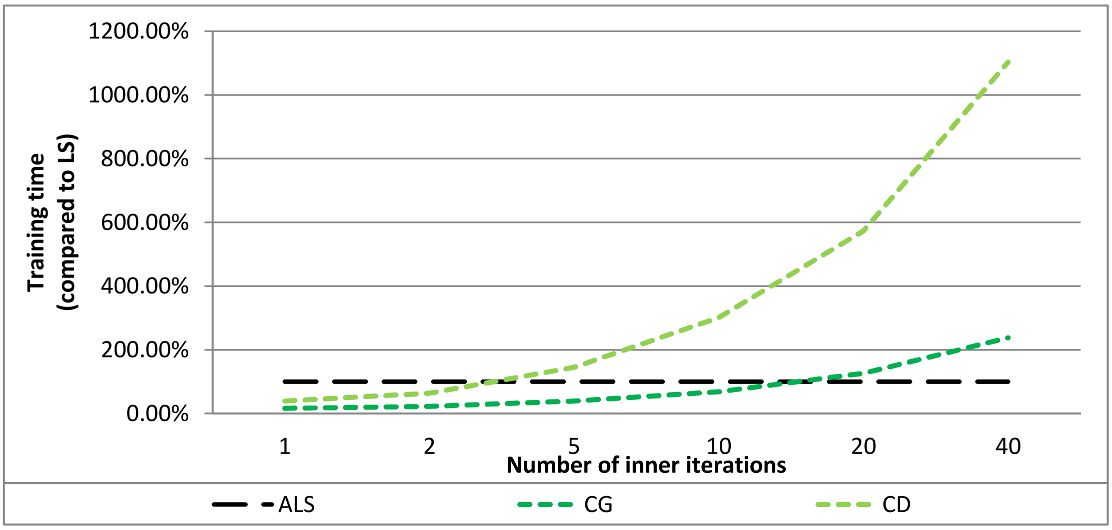

Figure 5.5: Running times with different number of inner iterations compared to that of the LS solver

The experiments show that CG converging to ALS can be either favorable or be a limit. On one hand it attests to the stability of the method, on the other hand, in some cases CD can outperform both ALS and CG. In practice, however, such a case is not typical (see Tables 5.2, 5.3 and 5.4), thus I conclude that the convergence of CG is useful.

Figure 5.5 compares the training times of CG and CD to ALS by different N_I values. The experiment used iTALSx, LastFM, sequentiality and $K = 80$. I note that the results are very similar with other settings.[7] CG scales significantly better with N_I than CD. CD reaches the training time of ALS around $N_I = 3 \dots 4$, that is only $\sim 4-5\%$ of K. CG reaches the training time of ALS much later, around $N_I = 15$, that is $\sim 19\%$ of K.

Approximate methods are used to speed up the training. Therefore such N_I should be used when the training time is significantly less. Our experiments suggest that this value is $1 \dots 2$ for CD and $1 \dots 10$ for CG if $K = 80$. For different values of K, these intervals change relative to K. Generally, $N_I = 1$ is a bad choice due to low accuracy (see Figure 5.4 for example), therefore N_I should be at least 2. Tables 5.2, 5.3 and 5.4 show that $N_I = 2$ is usually a good choice as the accuracy of ALS is usually well approximated. Larger N_I values are not advised for CD due to its poor scaling with N_I. For CG, N_I values up to 10–15% of K still result in considerable speed up, but usually small values $(2 \dots 5)$ are sufficient.

5.5.5 Convergence of accuracy

Figure 5.6 compares the accuracy of ALS, CG and CD (with 2 and 5 inner iterations each) after each recomputation of any feature matrix (i.e. their convergence w.r.t. accuracy). I investigated two cases: (1) when ALS converges faster (iTALSx, LastFM, sequentiality, $K = 80$), (2) when ALS converges more slowly (iTALS, VoD, sequentiality,

[7] In the following sense: N_I values relative to the number of features. That is, if K is lower/higher then approximate methods reach the training time of ALS at lower/higher N_I values.

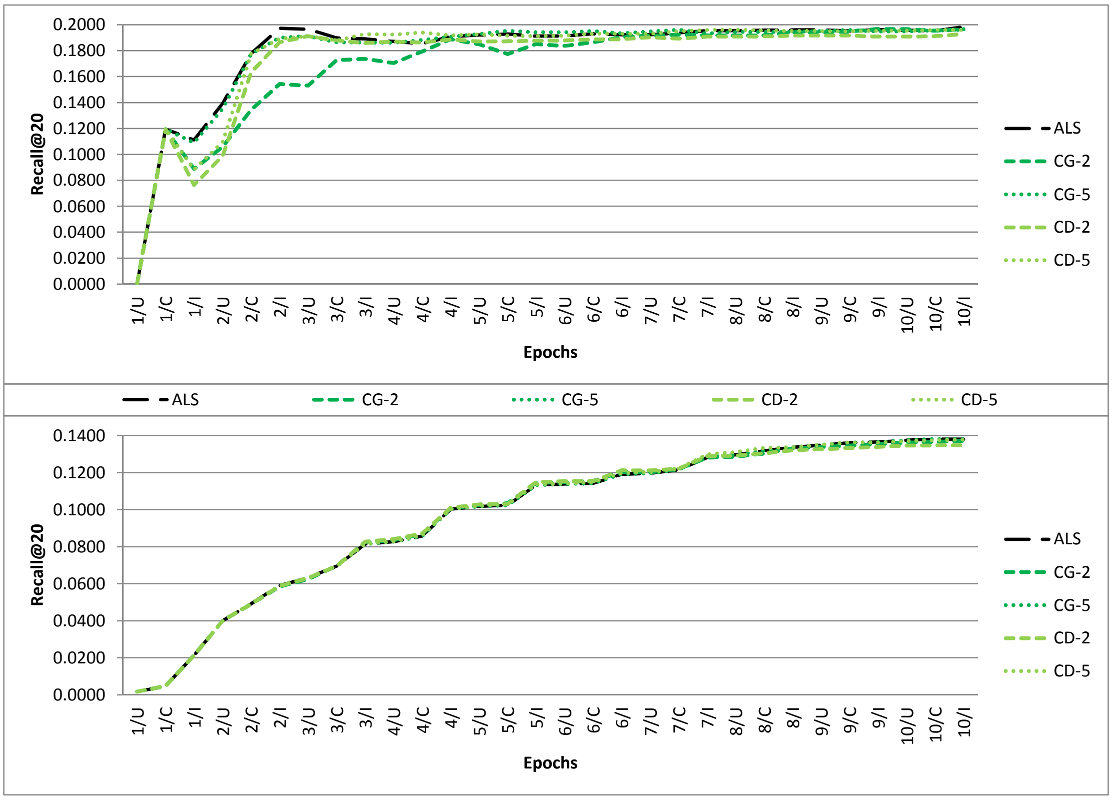

Figure 5.6: Accuracy of ALS, CG and CD methods (2 and 5 inner iterations) through epochs. The horizontal axis denotes how far is the algorithm in the computations. N/X denotes that the algorithm is in the N^{th} epoch and finished computing the X matrix ($X \in (U, I, C)$ as in User, Item and Context feature matrix). Top: case, where ALS converges quickly. Bottom: case, where ALS converges slowly.

$K = 40$). If ALS converges slowly then approximate methods can keep up and converge with basically the same speed. If the convergence of ALS is faster, approximate methods with $N_I = 2$ are initially less accurate, but achieve the same results after a few epochs. When the number of inner iterations is set higher ($N_I = 5$), approximate methods follow ALS quite nicely.

This suggests that $N_I = 5$ would be a better choice for CG as it follows ALS more closely. I suggest to use $N_I = 2$ if speed is important, because one mostly gets similar or better results than with ALS. If one prefers accuracy against training time then $N_I = 5$ (or higher) can be used. For CD I still suggest using $N_I = 2$ in every case, because of the fast increase of training time with larger N_I values.

5.5.6 Size of training data

In real life recommenders it is important to consider how much of the users' event history is to be used. Too much data does not only increase training time, but it may mask changes in taste and behavior. On the other hand, using only the recent events results in noisy training data and does not allow for models that capture long time interests. The optimal trade-off depends on the domain, the dataset itself and even the contexts

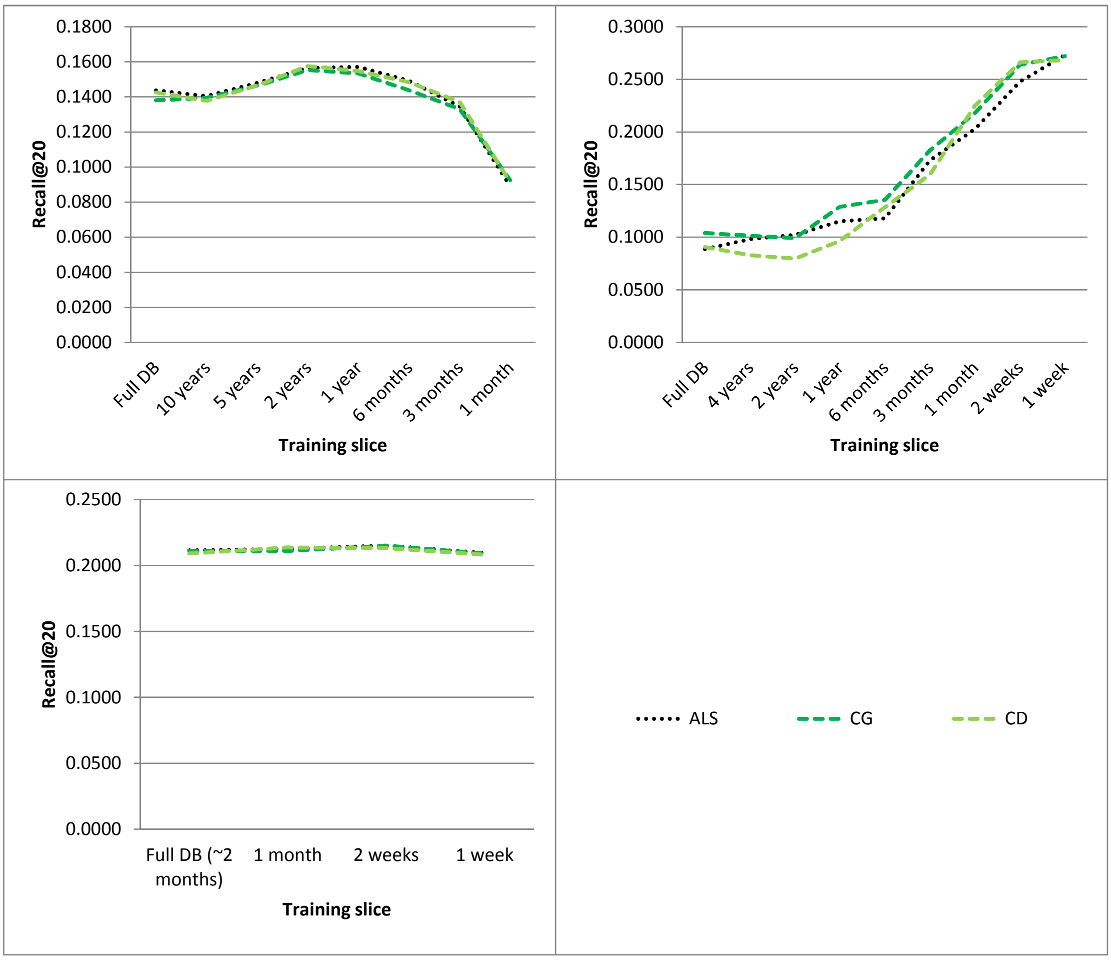

Figure 5.7: Accuracy of ALS, CG and CD methods (2 inner iterations) using different subsets of the training data. The end of the training interval remains the same, only the starting date changes. The three settings depicted here are Grocery, iTALS, sequentiality, 80 features (top left); LastFM, iTALS, seasonality, 40 features (top right); VoD, iTALS, sequentiality, 200 features (bottom left).

considered. Finding this optimal trade-off is beyond the scope of this research. Here I therefore only investigate if the approximate methods behave similarly to ALS.

Figure 5.7 shows the accuracy (w.r.t. recall@20) with using different slices of the training data in three different settings. Although the graph varies from setting to setting, both CG and CD follow ALS closely in all settings. Therefore I conclude that CG and CD behave similarly to ALS in this aspect as well.

5.6 Summary

In Chapter 5 I proposed ways to speed-up ALS learning through using approximate methods. The methods and the results described in this chapter were published in [25] and the following theses are based on them.

> **Thesis 5.1** *I proposed a general, conjugate gradient based approximation for ALS in ALS based factorization algorithms. I showed that this approximation scales linearly with the number of features in the range of practically used number of feature values. I showed that this allows the usage of higher factor models and finding better trade-offs between running time and accuracy. I showed that the recommendation accuracy is affected only in a minor way if the approximation is used instead of the exact ALS.*
>
> **Thesis 5.2** *I proposed a general, coordinate descent based approximation for ALS in ALS based factorization algorithms. I showed that this approximation scales linearly with the number of features in the range of practically used number of feature values. I showed that this allows the usage of higher factor models and finding better trade-offs between running time and accuracy. I showed that the recommendation accuracy is affected only in a minor way if the approximation is used instead of the exact ALS.*
>
> **Thesis 5.3** *I compared the conjugate gradient and coordinate descent based approximate solutions from a wide variety of aspects. I showed that the conjugate gradient based method is better, because it (a) follows the exact solution more closely in terms of recommendation accuracy; (b) is faster; (c) scales better; and (d) more stable.*
>
> **Thesis 5.4** *I determined a good trade-off between running time and recommendation accuracy for both approximate methods. I proposed to set the number of inner iterations to 2 in order to get this trade-off.*

Chapter 6

The General Factorization Framework

In this chapter I introduce the General Factorization Framework (GFF) [29]. GFF is a single flexible factorization algorithm that has no fixed preference model over the dimensions of its input. It rather takes the model as an input and takes care of the rest by computing the feature matrices.

6.1 Preference modeling easily

As I showed in Chapter 4 on iTALS and iTALSx, different preference models are appropriate for different situations. The only conclusion was that certain parameters of the factorization (e.g. number of features) and the dataset (e.g. sparsity) are beneficial for one or the other model. However it is worth noting that most factorization methods only use one of these two models (N-way, pairwise), although the number of possible models grows exponentially as the number of dimensions increases. It is also interesting to observe that both of these models are symmetrical, i.e. all dimensions fill the same role; meanwhile there are two distinguished dimensions in every recommendation task, the user and the item. (See Section 6.3 for more details.)

The preference model has an effect on the learning procedure as it was discussed in Chapter 4. It is especially problematic if transformations and separation of the computations are required in order to maintain low complexity. And this is exactly the case with the implicit feedback problem. For example iTALS and iTALSx seem very similar, but there are crucial steps that are different and even rely on different precomputed statistics.

The lack of proper exploration of preference modeling is due to the lack of flexible tools in which one can experiment with various models without being required to implement a specific algorithm for each model. I therefore created the General Factorization Framework (GFF), a single, flexible algorithm that takes the preference model as an input and computes latent feature matrices for the input dimensions. GFF allows us to easily experiment with various linear models on any context-aware recommendation task, be it explicit or implicit feedback based. GFF opens up a new research path in preference modeling under context.

The following properties were important at the design of GFF.

1. No restriction on context: GFF works on any context-aware recommendation problem independently of the number and the meaning of context dimensions.

2. Large preference model class: the only restriction on the preference model is that it must be linear in the dimensions of the problem[1]. This intuitive restriction does not restrict the applicability to real-world problems.

3. Data type independence: besides the practically more useful implicit case, explicit problems can be also addressed by simply changing the weighting scheme in the loss function.

4. Flexibility: the weighting scheme of GFF is very flexible, enabling to incorporate extra knowledge through the weights such time decay, as well as time dependent weighting, missing not at random hypotheses and more.

5. Scalability: GFF scales well both in terms of the number of interactions in the training set and in the number of features. This makes it applicable in real life recommender systems.

6.2 Basic GFF

GFF is a general modeling framework — inspired by the latent factor CF approach — which (1) efficiently integrates context data into the preference model; (2) allows experimentation with non-traditional models for more accurate preference estimation.

The basic framework relies on SA-MDM (see Section 1.2.1). In recommendation problems, the main goal is the modeling of user preferences on items, therefore one dimension is dedicated for the *users* and one dedicated for the *items*. I use one ID attribute in these dimensions. Other dimensions contain context data that helps modeling user preferences. Context can be the location or time of the interaction, the device on which the interaction was performed, or any other parameters that may influence the user preference, including weather, referral's link, search keyword, etc. Since SA-MDM is used, each context dimension contains exactly one attribute. The preference model is solely learnt from sample *events* (also called transactions).

Inspired by factorization methods, a feature vector of length K is assigned to each possible value of each attribute. These values are referred to as entities. For instance, the possible user IDs are entities. Therefore each attribute is represented as a feature matrix ($M^{(i)} \in \mathbb{R}^{K \times S_i}$, where S_i is the number of entities in the i^{th} dimension), assembled from the feature vectors of entities of the attribute. Since each dimension consists of exactly one attribute, dimensions are also represented by this feature matrix.

The modeling of the preferences is similar to the one used in Chapter 4. To briefly reiterate: SA-MDM compliant data can be arranged into an N_D dimensional tensor R. The values in the tensor are the preferences for the given combination of entities (i.e. a user-item-context combination). In case of explicit feedback data, the preferences are ratings. Typically the data space is very sparse, few ratings are observed, others are missing. The focus here is on the implicit case and therefore R is filled with binary

[1] Meaning that a dimension can not directly interact with itself in the model.

preference information: if a combination of entities occurred in the training data then the corresponding cell is set to 1, otherwise to 0.

$$r_{i_1,\dots,i_{N_D}} = \begin{cases} 1, & \text{if } t_{i_1,\dots,i_{N_D}} \in T \\ 0, & \text{otherwise} \end{cases} \tag{6.1}$$

Since the missing feedback is clearly a weaker signal of negative preference than the presence of positive feedback, the following weight function is used to weight entity combinations:

$$\mathcal{W} : (i_1, \dots, i_{N_D}) \to \mathbb{R}$$
$$\mathcal{W}(i_1, \dots, i_{N_D}) = \begin{cases} w^1(i_1, \dots, i_{N_D}) \gg w^0(i_1, \dots, i_{N_D}), & \text{if } t_{i_1,\dots,i_{N_D}} \in R \\ w^0(i_1, \dots, i_{N_D}) = \prod_{j=1}^{N_D} \left(\mu^{(j)} v_{i_j}^{(j)} + \gamma^{(j)}\right), & \text{otherwise} \end{cases} \tag{6.2}$$

where $w^1(i_1, \dots, i_{N_D})$ is the weight of entity combinations of the training set and $w^0(i_1, \dots, i_{N_D})$ is the weight of missing entity combinations. Both weight functions depend on the actual entities. This is the most permissive definition of the weight function that still allows for efficient computations. For the sake of simplicity, here I use a simple weight function by setting $\mu^{(j)} = 0$ and $\gamma^{(j)} = 1$ for all j, and setting $w^1(i_1, \dots, i_{N_D}) = \alpha \cdot \#(i_1, \dots, i_{N_D})$. That is $w^0(\cdot) = w_0 = 1$ for every entity combination and $w^1(\cdot)$ is proportional with the number of occurrences of the corresponding combination in the training set. This basic weighting assumes that entity combinations are missing at completely random [38] and that it is more important to accurately predict for entity combinations with actual feedback than for ones with no feedback. [2]

The simplified weight function used in the rest of this chapter is defined as follows:

$$\mathcal{W}(i_1, \dots, i_{N_D}) = \begin{cases} w^1(i_1, \dots, i_{N_D}) = \alpha \cdot \#(i_1, \dots, i_{N_D}) \gg w^0_{i_1,\dots,i_{N_D}}, & \text{if } t_{i_1,\dots,i_{N_D}} \in R \\ w^0(i_1, \dots, i_{N_D}) = w_0 = 1, & \text{otherwise} \end{cases} \tag{6.3}$$

The loss is defined as the weighted sum of squared loss:[3]

$$L = \sum_{i_1=1,\dots,i_{N_D}=1}^{S_1,\dots,S_{N_D}} \mathcal{W}(i_1, \dots, i_{N_D}) (\hat{r}_{i_1,\dots,i_{N_D}} - r_{i_1,\dots,i_{N_D}})^2 \tag{6.4}$$

The main novelty in GFF is that the preference model, i.e. the computation of $\hat{r}_{i_1,\dots,i_{N_D}}$ is an input of the algorithm. This allows us to experiment with *any linear models* beyond the usual ones. In the general framework, a preference model is a linear model of the feature vectors such that: (1) a model consists of sums of Hadamard (or elementwise) products; (2) each product contains at least two feature vectors; (3) in a product each feature vector belongs to a different attribute (linearity); (4) constant importance weights can be applied to each product.[4]

$$\hat{r}_{i_1,\dots,i_{N_D}} = 1^T \left(M_{\pi_1}^{(\sigma_1)} \circ \dots \circ M_{\pi_{p_1}}^{(\sigma_{p_1})} + \dots + M_{\pi_{p_{q-1}+1}}^{(\sigma_{p_{q-1}+1})} \circ \dots \circ M_{\pi_{p_q}}^{(\sigma_{p_q})}\right) \tag{6.5}$$

[2] Note that by setting $w^0 = 0$ and $w^1 = 1$ and using ratings in R we get the standard explicit setting in N_D dimensions.

[3] Regularization is omitted for clearer presentation, but ℓ_2 regularization is used in the actual algorithm.

[4] Omitted from the deduction for clearer presentation.

where $\sigma_k \in [1 \ldots N_D]$ and $\pi_k = i_j$ if $\sigma_k = j$. Biases can be included in the feature vectors and are not presented here separately due to clearer presentation. The model basically consists of selected interactions between members of a subset of dimensions.

6.2.1 Training with ALS(-CG)

Recall that the framework is designed to work also for implicit feedback, thus we need an optimization method that can efficiently handle the implicit setting. Methods that work for the explicit case can not be applied directly for the implicit case due to scalability issues that arise with the handling of missing feedback. One way to deal with this is by sampling the missing feedback, thus easily averting scalability issues. The other possibility is to smartly decompose computations into independently computable parts that can be shared through computations. I follow the latter route.

I use an Alternating Least Squares (ALS) method. In ALS only one matrix is updated at a time and all the other matrices are fixed. The optimization of the loss function is done through finding the optimal values in one feature matrix, given the others.

The two main advantages of ALS are (1) that it does not use sampling, therefore it is usually more accurate and converges faster; (2) the computations of the feature vectors – with linear models – are independent from each other and thus can be easily parellelized on multi-core or multiprocessor systems. The main problem with ALS is that it requires a least squares (LS) step for each feature vector computation and thus scales cubically in K that makes it hard to train high factor models. Therefore I approximate the solution of the least squares problem through conjugate gradient (CG) optimization introduced in Chapter 5. I derive the algorithm up to the point of efficiently computing the least squares problem. Then I apply CG to solve it. Chapter 5 on how to apply this learning strategy effectively.

I use the loss function from equation (6.4) and insert the general linear factorization model of equation (6.5) into it with the weighting scheme described in equation (6.3).

Without the loss of generality, I demonstrate the calculation of $M^{(i)}$ on the $M^{(1)}$ matrix. For clearer presentation, the members of the model (equation (6.5)) are grouped into two based on whether a column of $M^{(1)}$ is part of them:[5]

$$\begin{gathered}\hat{r}_{i_1,\ldots,i_{N_D}} = \\ = \underbrace{\left(M_{\pi_2}^{(\sigma_2)} \circ \ldots \circ M_{\pi_{p_1}}^{(\sigma_{p_1})} + \ldots + M_{\pi_{p_{k-1}+2}}^{(\sigma_{p_{k-1}+2})} \circ \ldots \circ M_{\pi_{p_k}}^{(\sigma_{p_k})}\right)^T}_{(\mathcal{Q}_1)^T} M_{i_1}^{(1)} + \\ + \underbrace{\left(M_{\pi_{p_k+1}}^{(\sigma_{p_k+1})} \circ \ldots \circ M_{\pi_{p_{k+1}}}^{(\sigma_{p_{k+1}})} + \ldots + M_{\pi_{p_{q-1}+1}}^{(\sigma_{p_{q-1}+1})} \circ \ldots \circ M_{\pi_{p_q}}^{(\sigma_{p_q})}\right)^T}_{(\mathcal{Q}_2)^T} 1\end{gathered} \tag{6.6}$$

When recomputing $M^{(1)}$, every other matrix is fixed, thus L is convex in the elements of $M^{(1)}$. The minimum is reached when $\partial L / \partial M^{(1)}$ is zero. The columns of $M^{(1)}$ can be computed separately, because the derivative is linear in them. Each column is computed

[5] To avoid more complex notation, we assume that the columns of $M^{(1)}$ are the first members in the products where they are present.

similarly, therefore only the steps for $M_1^{(1)}$ (the first column of $M^{(1)}$) are shown:

$$\begin{aligned}\frac{\partial L}{\partial M_1^{(1)}} = & \underbrace{-2 \sum_{i_2=1,\ldots,i_{N_D}=1}^{S_2,\ldots,S_{N_D}} r_{1,i_2,\ldots,i_{N_D}} \mathcal{W}(1,i_2,\ldots,i_{N_D})\mathcal{Q}_1 +}_{\mathcal{O}} \\ & \underbrace{+2 \sum_{i_2=1,\ldots,i_{N_D}=1}^{S_2,\ldots,S_{N_D}} w_0 \hat{r}_{1,i_2,\ldots,i_{N_D}} \mathcal{Q}_1 +}_{\mathcal{I}_2=\mathcal{I}+\mathcal{J}M_1^{(1)}} \\ & \underbrace{+2 \sum_{i_2=1,\ldots,i_{N_D}=1}^{S_2,\ldots,S_{N_D}} (\mathcal{W}(1,i_2,\ldots,i_{N_D}) - w_0) \hat{r}_{1,i_2,\ldots,i_{N_D}} \mathcal{Q}_1}_{\mathcal{I}_1=\mathcal{I}'+\mathcal{J}'M_1^{(1)}}\end{aligned} \tag{6.7}$$

I introduce $\mathcal{O}$, $\mathcal{I}_1 = \mathcal{I}' + \mathcal{J}' M_1^{(1)}$ and $\mathcal{I}_2 = \mathcal{I} + \mathcal{J} M_1^{(1)}$ to simplify further equations. $\mathcal{O}$ is the weighted sum of $\mathcal{Q}_1$ type vectors from equation (6.6) over all possible configurations involving the first entity of the first dimension. The weights are the products of corresponding elements of the preference tensor R and the value of the weighting function $\mathcal{W}$ for that setting. Due to the values of the preferences, most of the members of this sum are zero. Both $\mathcal{I}_1$ and $\mathcal{I}_2$ are the sum of a coefficient matrix multiplied by the vector we seek (i.e. $M_1^{(1)}$ in this case) and a vector. The difference is that these parts of $\mathcal{I}_2$ (i.e. $\mathcal{I}$ and $\mathcal{J}$) are the same for every column of $M^{(1)}$ (and therefore can be precomputed); while those of $\mathcal{I}_1$ (i.e. $\mathcal{I}'$ and $\mathcal{J}'$) are not.

$\mathcal{O}$, $\mathcal{I}'$ and $\mathcal{J}'$ can be computed efficiently (see section 6.2.2), however the naive computation of $\mathcal{I}$ and $\mathcal{J}$ is expensive. Therefore we further transform $\mathcal{I}_2$. With the expansion of $\hat{r}_{1,\ldots,i_{N_D}}$ (substituting (6.6) with $i_1 = 1$):

$$\mathcal{I}_2 = 2w_0 \sum_{i_2=1,\ldots,i_{N_D}=1}^{S_2,\ldots,S_{N_D}} \mathcal{Q}_1(\mathcal{Q}_1)^T M_1^{(1)} + \mathcal{Q}_1(\mathcal{Q}_2)^T 1 \tag{6.8}$$

Expanding either $\mathcal{Q}_1(\mathcal{Q}_1)^T$ or $\mathcal{Q}_1(\mathcal{Q}_2)^T$ results in sums of matrix products, where the arguments are the elementwise products of multiple feature vectors:

$$\sum_{i_2=1,\ldots,i_{N_D}=1}^{S_2,\ldots,S_{N_D}} \left(M_{i_{j_1}}^{(j_1)} \circ \ldots \circ M_{i_{j_m}}^{(j_m)}\right) \left(M_{i_{l_1}}^{(l_1)} \circ \ldots \circ M_{i_{l_t}}^{(l_t)}\right)^T \tag{6.9}$$

where $j_i \neq j_k$ if $i \neq k$, $l_i \neq l_k$ if $i \neq k$, $j_i \in [2 \ldots n]$ and $l_k \in [2 \ldots n]$. With rearranging this expression, only the following types of quantities are needed to be computed:

$$\begin{aligned} &\text{(a)}\ C^{(j)} = \sum_{i=1}^{S_j} M_i^{(j)} \left(M_i^{(j)}\right)^T, \\ &\text{(b)}\ O^{(l)} = \sum_{i=1}^{S_l} M_i^{(l)}, \\ &\text{(c)}\ S_k, \end{aligned} \tag{6.10}$$

where (a) $C^{(j)} \in \mathbb{R}^{K \times K}$ is the covariance matrix of the feature vectors of the j^{th} feature matrix; (b) $O^{(l)} \in \mathbb{R}^K$ is the sum of the feature vectors of the l^{th} feature matrix; (c) $S_k \in \mathbb{R}$ is the domain size. (6.9) can be computed from (a), (b) and (c) using the following operations: (1) elementwise product of $\mathbb{R}^{K \times K}$ matrices; (2) elementwise product of $\mathbb{R}^K$ vectors; (3) matrix product of $\mathbb{R}^K$ vectors; (4) matrix–scalar multiplication. Note that S_k is a fix value during the training process, and $C^{(j)}$ and $O^{(j)}$ only changes after the j^{th} feature matrix is recomputed. Therefore these quantities can be precomputed and should be updated only once per epoch.

After $\mathcal{O}$, $\mathcal{I}'$, $\mathcal{J}'$, $\mathcal{I}$ and $\mathcal{J}$ from equation (6.7) are computed, $\frac{\partial L}{\partial M_1^{(1)}} = 0$ can be solved for $M_1^{(1)}$. Instead of the least squares solver (LS), I use an approximate conjugate gradient solver to get the new value of the feature vector. Algorithm 6.2.1 shows the high level pseudocode of the training.

Algorithm 6.2.1 ALS-based learning of the general framework on implicit data

Input: T: training data; MODEL: the description of the desired model K: number of features; E: number of epochs; λ: regularization coefficient

Output: $\{M^{(i)}\}_{i=1,\dots,N_D}$ $K \times S_i$ sized low rank matrices

```
procedure Train($T$, MODEL, $K$, $E$, $\lambda$)
for $i = 1, \dots, N_D$ do
    $M^{(i)} \leftarrow$ Random $K \times S_i$ sized matrix
    $C^{(i)} \leftarrow \sum_{k=1}^{S_i} M_k^{(i)} \left(M_k^{(i)}\right)^T$ and $O^{(i)} \leftarrow \sum_{k=1}^{S_i} M_k^{(i)}$
end for
for $e = 1, \dots, E$ do
    for $i = 1, \dots, N_D$ do
        Compute the shared parts $\mathcal{I}$ and $\mathcal{J}$
        for $j = 1, \dots, S_i$ do
            Compute $\mathcal{O}$, $\mathcal{I}'$ and $\mathcal{J}'$
            Add regularization
            Solve $\frac{\partial L}{\partial M_j^{(i)}} = 0$ for $M_j^{(i)}$
        end for
        $C^{(i)} \leftarrow \sum_{k=1}^{S_i} M_k^{(i)} \left(M_k^{(i)}\right)^T$ and $O^{(i)} \leftarrow \sum_{k=1}^{S_i} M_k^{(i)}$
    end for
end for
return $\{M^{(i)}\}_{i=1,\dots,N_D}$
end procedure
```

So far regularization and biases were neglected. Regularization can be done by adding a $K \times K$ sized diagonal matrix to $\mathcal{J} + \mathcal{J}'$ (i.e. to the coefficient matrix of $M_j^{(i)}$) just before computing the feature vector. The model (6.5) can be extended with biases by adding $\sum_{i=1}^{N_D} \sum_{j=1}^{S_i} v_{i,j} b_{i,j}$ to it, where $b_{i,j}$ is the bias value for the j^{th} entity of the i^{th} attribute and $v_{i,j}$ is the weight of the bias. The training of this biased model can also be done efficiently with the same complexity as the non-biased $(K + 1)$-feature model.

6.2.2 Complexity of training

The complexity of one epoch (i.e. computing each matrix once) is $O(N_D N^+|O|K^2 + \sum_{i=1}^{N_D} S_i K^3)$ with a naive LS solver (see Table 6.1 for breakdown). This is reduced to $O(N_D N^+|O|K + \sum_{i=1}^{N_D} S_i K^2)$ with a carefully implemented CG solver. Since $|O|N_D N^+ \gg \sum_{i=1}^{N_D} S_i$ and K is small ($K \in [20\dots300]$), the first term dominates. Therefore the algorithm scales *linearly* with both the number of transactions and K in practice (see Section 6.3.4 for empirical results on running times).

Table 6.1: Complexity of computations

Task	Complexity	Comments
Computations required per columns of $M^{(1)}$		
$\mathcal{O}$, $\mathcal{I}'$ and $\mathcal{J}'$	$O(N_1^+ K^2 \vert O\vert)$	N_1^+ is the number of training events, where the value of the $A^{(1)}$ attribute is $a_1^{(1)}$, and $\vert O\vert$ is the complexity of the model (i.e. the number of vector operations to compute $\hat{r}$). This is possible due to the definition of c weights and r preferences, as most of the members in the sums of $\mathcal{O}$, $\mathcal{I}'$ and $\mathcal{J}'$ are in fact zeroes.
Solving for $M^{(1)}$	$O(K^3)$	Using the naive LS solver.
Total complexity of the above for all columns of $M^{(1)}$: $O(N^+K^2\vert O\vert + S_1K^3)$, *($N^+$ is the number of transactions)*		
Computations once per computing $M^{(1)}$		
Computing $\mathcal{I}$ and $\mathcal{J}$	$O(\vert O\vert K^2)$	Assembled from members described in equation (6.10): $C^{(j)}$ and $O^{(j)}$. These need to be recomputed when $M^{(j)}$ changes.
Recomputing $C^{(1)}$ and $O^{(1)}$	$O(S_1 K^2)$	Computed after finishing the recomputation of $M^{(1)}$.
Total complexity of an epoch: $O(N_D N^+\vert O\vert K^2 + \sum_{i=1}^{N_D} S_i K^3)$		

6.2.3 Special cases

I now show that standard factorization algorithms are special cases of GFF. In standard 2D MF for implicit feedback [30], the preference of user u on item i is predicted as product of user and an item features: $\hat{r}_{u,i} = 1^T\left(M_u^{(1)} \circ M_i^{(2)}\right)$. iTALS – the context-aware tensor factorization model [24] – with 3 dimensions predicts the preference of user u on item i under context-state c as $\hat{r}_{u,i,c} = 1^T\left(M_u^{(1)} \circ M_i^{(2)} \circ M_c^{(3)}\right)$, the product of each features. Its modification – iTALSx – does the same by using $\hat{r}_{u,i,c} = 1^T\left(M_u^{(1)} \circ M_i^{(2)}\right) + 1^T\left(M_i^{(2)} \circ M_c^{(3)}\right) + 1^T\left(M_i^{(2)} \circ M_c^{(3)}\right)$. If ratings are used in R with $w^0(\cdot) = 0$, $w^1(\cdot) = 1$ and $\hat{r}_{u,i} = 1^T\left(M_u^{(1)} \circ M_i^{(2)}\right)$, we get the classic ALS MF algorithm [7]. SVD++ [34] can be also derived from GFF if explicit compliant weighting and preferences are used and the items rated by the users are included through binary attributes. The model should be set accordingly to the SVD++ model. However I recommend using the extended framework (see Section 6.5) instead of using many binary attributes, because of the increased training time.

6.3 Modeling preferences under context

In this section I demonstrate the usefulness of the GFF by examining several models therein. Recall that the preference model is an input of the framework that allows experimentation with novel models without implementing a specific algorithm. Therefore novel models can be examined and compared to the traditional N-way and pairwise interaction models.

A context-aware problem

My aim is to apply GFF to the implicit feedback based context-aware recommendation problem and find models that generally perform well. The area of context-aware problems is wide, as any additional information to the user–item interaction can be considered as context. In compliance with SA-MDM, we assume that the context dimensions are event contexts, meaning that their value is not determined solely by the user or the item; rather it is bound to the transaction. E.g. the time of the transaction is an event context, while the genres of the item is not.

Due to practical considerations, the context dimensions I use can be easily derived from transactional data using the timestamp of the events. I use the time of the transactions to derive seasonality and the order of the events to derive sequentiality. (See more detailed definitions in Section 1.4.2.) The importance of these context dimensions is that they are always available to use when timestamps are available, which is always recorded during implicit feedback collection. Using these domain independent contexts that require no additional data collection can increase recommendation accuracy when used with the proper preference model.

6.3.1 Preference models

First, I introduce a highly simplified notation for preference models. The four dimensions are denoted by U, I, S and Q for users, items, seasonality and sequentiality respectively. The models consist of selected interactions between selected dimensions. An interaction is denoted by putting the dimensions after one another. E.g. UI is the user–item interaction, USI is the user–item–seasonality interaction and so on. A model usually contains more than one interaction. Table 6.2 shows examples of this notation.

There are 11 different possible interactions with 4 dimensions, therefore the number of possible preference models is $2^{11}-1 = 2047$. Removing the ones that do not contain U or I, we still get 2018 potential models. In the field of context-aware recommendations, state-of-the-art factorization methods use two models. The pairwise interaction model ($UI + US + IS + UQ + IQ + SQ$ with all 4 dimensions) ([23, 51, 52]) assumes pairwise interaction between each pair of dimensions. On the other hand the N-way model ($UISQ$ with all 4 dimensions) ([24, 63]) assumes that the preferences can be best described by

Table 6.2: Examples of the simplified notation system

UI	Vanilla MF model (user–item interactions): $\hat{r}_{u,i} = 1^T \left(M_u^{(U)} \circ M_i^{(I)} \right)$
$USQI$	N-way model with all 4 dimensions (tensor factorization): $\hat{r}_{u,i,s,q} = 1^T \left(M_u^{(U)} \circ M_i^{(I)} \circ M_s^{(S)} \circ M_q^{(Q)} \right)$
$UI + US + IS$	Pairwise interaction model with 3 dimensions (U, I, S): $\hat{r}_{u,i,s} = 1^T \left(M_u^{(U)} \circ M_i^{(I)} + M_u^{(U)} \circ M_s^{(S)} + M_i^{(I)} \circ M_s^{(S)} \right)$

the joint interaction of all dimensions. [51] also mentions the generalization of the pairwise interaction model, coined d-way interaction model (e.g. the 3-way interaction model: $UI{+}US{+}IS{+}UQ{+}IQ{+}SQ{+}USI{+}UQI{+}USQ{+}ISQ$ with all 4 dimensions). This model includes all interactions between subsets of dimensions up to d size. The authors argue that such a model is slow to train and usually does not result in more accurate recommendations.

I approach the preference modeling from the perspective of the context-aware recommendation task. In this setting the users initiate transactions with the items. Additional variables (context) may or may not influence user behavior, therefore not all possible interactions should be considered for preference modeling. I focus on the ones where either the user, the item or both interact with a context. Interactions where contexts interact with each other are disregarded (see Section 6.3.3 for additional justification), except for SQ that is only kept for compatibility's sake with the pairwise model. Therefore we get to the followings:

- ***UI*:** Interaction between users and items, the classic CF model.
- ***USI*, *UQI*, *USQI*:** The context value dependent reweighting of the user–item relation, i.e. the context influences how the users interact with items. More context dimensions can be used for reweighting. But the more we use, the more sensitive it becomes to noise and more latent features are required for filtering this out [23].
- ***US*, *UQ*:** The user–context interaction produces a context dependent user bias that does not play role during the ranking but has noise filtering properties during training. We allow only one context in these interactions, because additional contexts would assume that different context dimensions interact somehow.
- ***IS*, *IQ*:** The item–context interaction results in a context dependent item bias that helps in ranking as well as in learning. Only one context is allowed in these interactions.
- ***SQ*:** Interactions between the two context dimensions. Required for the traditional pairwise model.

The models I used are depicted on Figure 6.1. Models on the right side follow the pairwise interaction scheme, while models on the left are of the N-way flavor. *Traditional models* – that were used also earlier – are indicated with orange background and black text and *novel models* are with green background and white text. The models are sorted to layers based on the dimensions used. In 2D there is only the classical UI model of CF. With the inclusion of one context dimension (either S or Q) the N-way and the

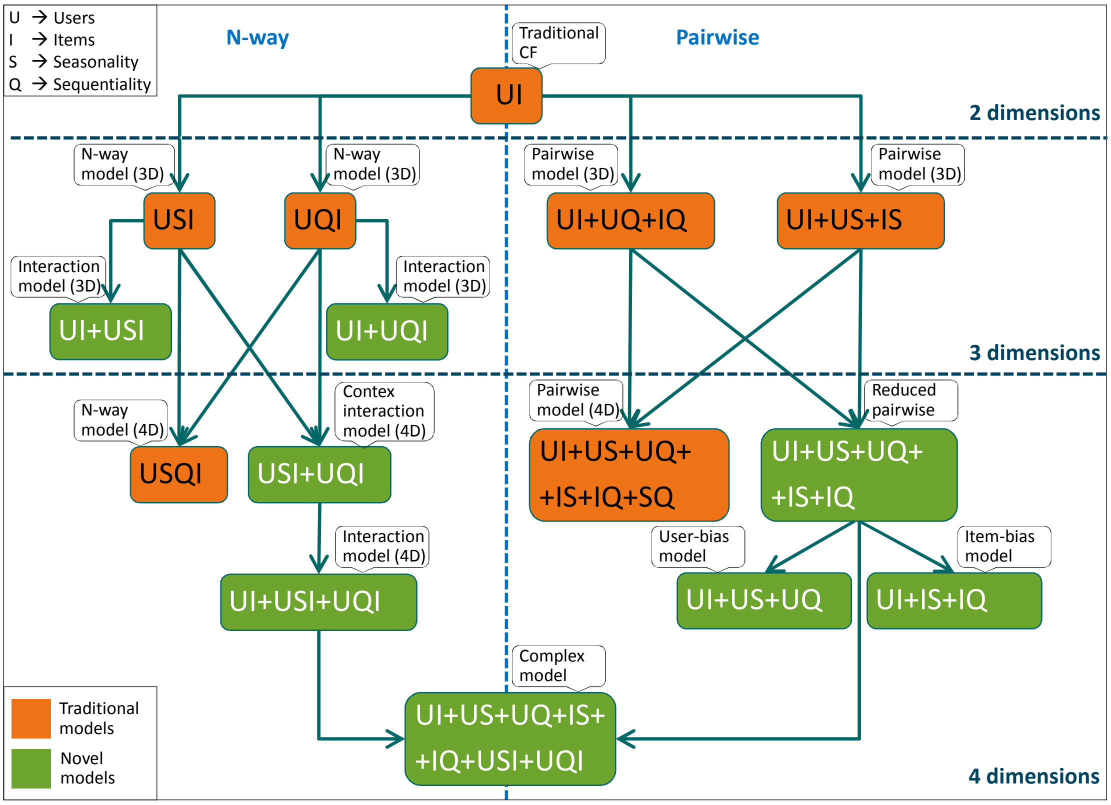

Figure 6.1: Hierarchy of the models

pairwise philosophy of preference modeling diverges. There are only a few novel models with three dimensions and I only selected those that I coined *interaction model*. Things get interesting with all four dimensions where one can create many novel models. I selected the following novel models for experimentation:

- **Interaction model ($UI + USI + UQI$):** This model is the composite of the base behavior of the users (UI) and their context-influenced modification of this behavior (USI and UQI). This model assumes that the preferences of the users can be divided into context independent and dependent parts. In the latter the user–item relation is reweighted by a context dependent weight vector. $USQI$ is not included due to the noisiness of reweighting by more than one weight vector simultaneously.
- **Context interaction model ($USI+UQI$):** Preferences in this model are modeled by solely context dependent parts, i.e. it assumes that user–item interactions strongly depend on the context and this dependency affects the whole interaction rather than solely the items or users.
- **Reduced pairwise model ($UI+US+IS+UQ+IQ$):** This model is a minor variation of the traditional pairwise model with the exclusion of the interaction between context dimensions (SQ). The interaction with context is done separately by users and items, i.e. it does not affect the whole user–item relation.
- **User bias model ($UI + US + UQ$):** Here it is assumed that only the user interacts with the other dimensions. This results in a model where the user–

item relation is supported by context dependent user biases. Note that during recommendation, these context-aware user biases are constant, thus do not affect the ranking. However they might filter out some context related noise during training.

- **Item bias model ($UI+IS+IQ$):** This model assumes that the effect of context can be described by context dependent item biases (e.g. items are popular under certain conditions). The item biases affect the ranking as well as filter context related noise during training.
- **A complex model ($UI+US+IS+UQ+IQ+USI+UQI$):** This model is the composite of the reduced pairwise and the interaction model. It can be also treated as a reduced 3-way interaction model from which the context-context interactions are omitted.

Note that I restricted our model space to those where exactly one feature matrix belongs to each dimension. In GFF it is possible to use several set of features for selected dimensions. By doing so it is possible to decouple the modeling of different effects from each other. For example user and item interaction with a certain context dimension can be modeled separately by using two sets of feature for the context dimension. This is a far reaching research direction that is out of the scope of in this experiment, but nonetheless made available by GFF.

6.3.2 Results

Table 6.3: Recall@20 values for different models within the framework. Differences between the performance of models are statistically significant at $p = 0.05$. Traditional models are with gray background. Best results are typeset bold.

Model	Grocery	TV1	TV2	LastFM	VoD
$USI + UQI$ (context interaction model)	0.1504	**0.1551**	0.2916	0.1984	0.1493
$UI + USI + UQI$ (interaction model)	**0.1669**	0.1482	**0.3027**	**0.2142**	**0.1509**
$USQI$ (N-way model)	0.1390	0.1315	0.2009	0.1906	0.1268
$UI + US + IS + UQ + IQ$ (reduced pairwise model)	0.1390	0.1352	0.2388	0.1884	0.0569
$UI + US + UQ$ (user bias model)	0.1619	0.0903	0.1399	0.1993	0.0335
$UI + IS + IQ$ (item bias model)	0.1364	0.1266	0.2819	0.1871	0.1084
$UI{+}US{+}IS{+}UQ{+}IQ{+}SQ$ (pairwise interaction model)	0.1388	0.1344	0.2323	0.1873	0.0497
$UI + US + IS + UQ + IQ +$ $USI + UQI$ (complex model example)	0.1389	0.1352	0.2427	0.1866	0.0558

Table 6.3 shows the accuracy in terms of recall@20 of two traditional models and the six novel models I've just introduced.

There exists a novel model with all five datasets that performs better than the traditional models. In 4 out of 5 cases, the interaction model ($UI + USI + UQI$) is the best and it is the second best in the remaining one case. Thus this model is not only intuitively sound but also performs well, which underpins our assumptions on preference modeling. The context interaction model ($USI + UQI$) comes second in 3, and third in 2 cases. Interestingly, the user bias model ($UI + US + UQ$) is the second best in 2 out of 5 cases while worst one in the other 3 cases. This can be explained by the differences between the repetitiveness of the datasets. Highly repetitive datasets are affected more heavily by sequentiality and benefit from the noise filtering property of the UQ member. As sequentiality is more closely related to user behavior than to the items, UQ is much more effective than IQ.

The reduced pairwise model is better than the full pairwise interaction model in all cases, however the difference is negligible in 3 out of 5 cases. But the difference is $\sim$ 14.44% by VoD and $\sim$ 2.8% by TV2 dataset. Finally, note that the complex model generally does not improve over the reduced pairwise model considerably and is always worse than the interaction and the context interaction models. Three way interactions contribute to the score in a lesser way, because feature values are generally small and thus three way products generally give smaller values. This causes the context dependent biases to be more prominent initially, thus the features are set accordingly to optimize the bias values. This confirms the observations by [51] finding the d-way interaction model no more useful than the pairwise interaction model. However this problem might be tackled by using two sets of features for S and Q, separately for the three way interactions and context dependent biases.

Table 6.4: Improvements over traditional models

Dataset	**Best model**	**Improvement**	**Models better than traditional ones (out of 6)**
Grocery	$UI + USI + UQI$	+20.14%	3
TV1	$USI + UQI$	+15.37%	2
TV2	$UI + USI + UQI$	+30.30%	5
LastFM	$UI + USI + UQI$	+12.40%	3
VoD	$UI + USI + UQI$	+19.02%	2

Table 6.4 summarizes the improvements by novel models over the traditional ones. The best novel model (interaction model in 4/5 and context interaction model 1/5) outperforms the best traditional model by 12–30% in terms of recall@20. This difference is significant. Besides, there are several novel models for each dataset that outperform the traditional models by more than 5%. These include the context interaction model and models specifically good for the data (e.g. the user bias model for Grocery and LastFM).

So far the number of features was fixed at $K = 80$. However this parameter can significantly affect the relation of models. In Section 4.4 I compared the pairwise and the N-way model on two 3 dimensional context-aware problems. I found that pairwise models perform better with lower number of factors, but N-way models improve more rapidly as K increases. This is due to low factor models blurring different aspects of the

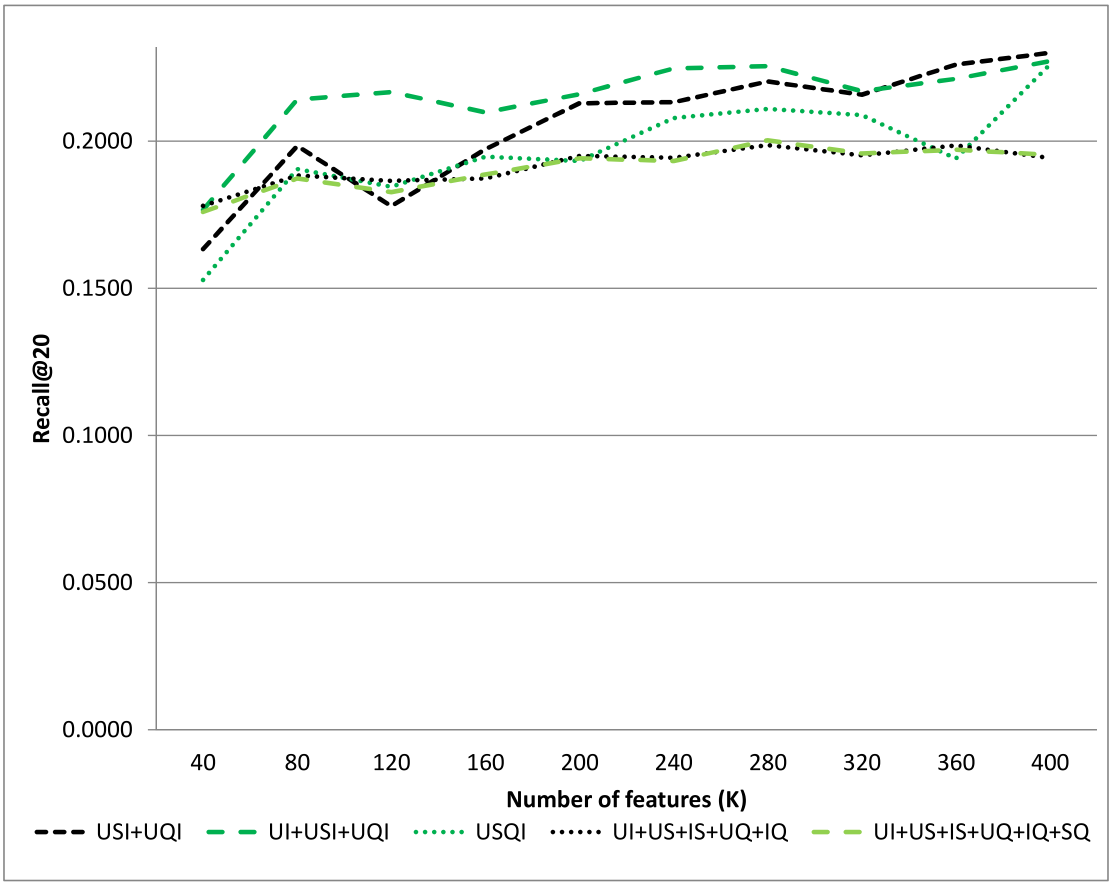

Figure 6.2: Model accuracy versus number of factors.

entities together, thus making the reweighting of the N-way model more difficult, if not impossible.

Figure 6.2 depicts recall@20 for different values of K ranging from 40 to 400. Five models were selected for this experiment: the well performing interaction ($UI + USI + UQI$) and context interaction ($USI + UQI$) model; the traditional N-way ($USQI$) and pairwise ($UI + US + IS + UQ + IQ + SQ$) model; and the reduced pairwise model ($UI + US + IS + UQ + IQ$). The results are presented on the LastFM dataset. At $K = 40$ $USI + UQI$ and $USQI$ are clearly worse than the other models and the reduced pairwise model is even slightly better than $UI + USI + UQI$ and the pairwise model. By $K = 400$ the context interaction model is leading slightly (within 2%) compared to the interaction and the traditional N-way models. The pairwise and reduced pairwise models on the other hand lag behind by more than 15%. We can observe that as K increases, the accuracy of models with members of higher order of interactions increase more rapidly. The N-way model improves the fastest and would probably outperform other models if the K is sufficiently high. On the other hand, larger K values (at or beyond 400) require longer training time and even more importantly, comes with longer recommendation times, therefore their practical use is limited.

It is also worth noting that $UI + USI + UQI$ performs more stable than $USI + UQI$ or the N-way model. This is due to the UI part (i.e. the context independent user–

item relation) stabilizing the prediction. I conclude that the interaction model ($UI + USI + UQI$) performs well not just for $K = 80$ but for practically important K values in general.

6.3.3 Context-context interactions

We discussed that interactions between context dimensions should be excluded from the model. Comparing the pairwise and reduced pairwise model showed that modeling such interactions does not increase accuracy and sometimes even degrades it. From a recommendation task perspective, context dimensions never interact with each other. They can influence the users' behavior (also via their active/inactive status), and through the users they affect the consumption pattern of items as well. One could argue that other context dimensions are also affected in a similar way. However recall that not all dimensions are equal and the main focus in recommendation is to recommend items to the users (under different contexts). Even if certain context dimensions correlate, there is no direct interaction between them.

I also argue that context dimensions should be independent from each other. The context-aware recommendation task becomes harder [45] and slower [55] as the number of dimensions increase. Therefore context dimensions should ideally capture different aspects of the data rather than describing the same or highly correlated characteristics in different ways.

The context dimensions of the example setting (S and Q) are fairly independent from each other. To quantify the independence of two context dimensions $C^{(1)}$ and $C^{(2)}$, the following probability distributions can be approximated from the training data: $P(C^{(1)}) = \left\{P(C^{(1)} = c_i^{(1)})\right\}$ and $P_j(C^{(1)}) = \left\{P(C^{(1)} = c_i^{(1)}|C^{(2)} = c_j^{(2)})\right\}$. The average Kullback–Leibler divergence between $P(C^{(1)})$ and $P_j(C^{(1)})$ for all j can be then computed. Small average KL divergence means that $P(C^{(1)})$ can be used in the place of $P_j(C^{(1)})$ distributions. In other words knowing the state in $C^{(2)}$ gives us low information on the state in $C^{(1)}$.

This experiment was executed with $C^{(1)} = C^{(2)} = S$ (totally dependent context dimensions); $C^{(1)} = S$, $C^{(2)} = Q$ (sequentiality's information on seasonality); $C^{(1)} = Q$, $C^{(2)} = S$ (seasonality's information on sequentiality); $C^{(1)} = S$, $C^{(2)} = S'$; $C^{(1)} = S'$, $C^{(2)} = S$, where S' is seasonality with the same season as S, but uses different time bands. The results are shown in table 6.5. It is obvious that seasonality has little information on sequentiality and vice versa, therefore these context dimensions hardly correlate. This explains why the full pairwise model performs worse than the reduced pairwise model.

6.3.4 Training time

Figure 6.3 shows the time of one epoch (i.e. computing each feature matrix once) for selected models for different values of K on the VoD dataset. The experiments were carried out using a single core of a multi-core CPU machine. Note that the computation can be easily parallelized, therefore these training times can be greatly reduced in practice. As stated in Section 6.2.2, the running time scales linearly with the number of features for K in the practically useful range. There is a difference between the actual time of training for different models as it also depends on the complexity of the model.

Table 6.5: Average KL divergences from $P(C^{(1)})$ to $P_j(C^{(1)})$

Data set	**Average** $D_{KL}\left(P_j(C^{(1)})\|\|P(C^{(1)})\right)$				
	$C^{(1)}=S$ $C^{(2)}=S$	$C^{(1)}=S$ $C^{(2)}=Q$	$C^{(1)}=S$ $C^{(2)}=S'$	$C^{(1)}=Q$ $C^{(2)}=S$	$C^{(1)}=S'$ $C^{(2)}=S$
Grocery	3.2574	0.0696	2.2997	0.0695	2.5238
TV1	3.1032	0.0189	1.7235	0.0171	1.5203
TV2	2.8132	0.0811	2.7979	0.0947	2.7707
LastFM	2.6376	0.0030	2.6162	0.0976	2.5618
VoD	2.6300	0.0262	1.8024	0.0547	2.1650

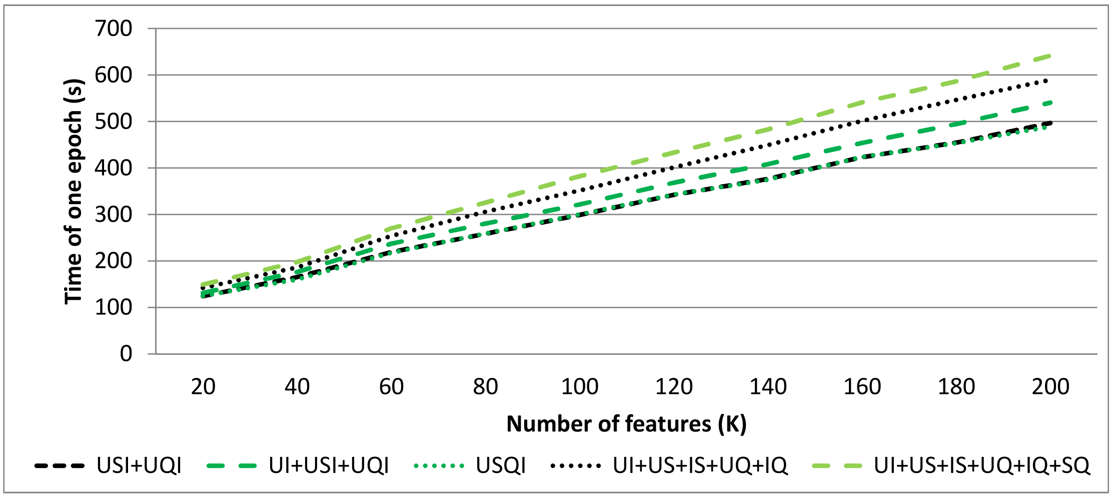

Figure 6.3: Training times of models

The complexity of the model is the number of operations required to compute the preference model. If the set of dimensions is fixed, the scaling in the model complexity is linear. In accordance with this the N-way model is the fastest and the pairwise model is the slowest from the selected ones. Also note that modeling the useless SQ interaction also slows down the training.

6.4 Comparison with state-of-the-art algorithms

In this section I compare GFF with other methods. The qualitative comparison focuses on pointing out key differences between GFF and other factorization algorithms. Although the main advantage of GFF is not necessarily that it can outperform other methods, but rather its flexibility (regarding the model and weighting); I also include a quantitative comparison with widely accepted algorithms such as Factorization Machines (FM) [51] and Bayesian Personalized Ranking (BPR) [53].

6.4.1 Qualitative comparison with factorization methods

Factorization Machines

Rendle et. al [51] proposed factorization machines (FM) as a general factorization method. FM is for rating prediction (explicit flavor). Implicit feedback problem can be tackled through subsampling negative feedback. Each rating is associated with different attributes, for example the user who rated, the item that was rated, the context of the rating, metadata of the item, etc. The preference model is a full (weighted) pairwise model: the prediction score is given by the sum of pairwise interaction scores between every pair of dimensions.[6] The weight of a certain interaction is determined by the weight of the two corresponding attributes; this is an input of the algorithm. It builds on the SA-MDM datamodel just like basic GFF, therefore it handles composite dimensions through binary variables as dimensions. This solution has two drawbacks: (1) it significantly increases the training time; (2) and a lot of unnecessary interactions are modeled between these binary attributes. The authors proposed a partitioning method to overcome this problem in [55], which basically results in excluding certain interactions from the pairwise model. The latent feature vectors can be learned by several learning methods: stochastic gradient descent (SGD), coordinate descent[7], adaptive SGD and a Bayesian inference using Markov Chain Monte Carlo (MCMC). The latter is advised as the best one of the four. The implementation of FM is available in libFM.[8]

The key differences between GFF and FM are as follows: (1) FM uses a subset of all possible pairwise interactions between dimensions, while GFF can use arbitrary linear preference model. (2) FM handles implicit feedback through subsampling the missing (negative) feedback and is mainly an explicit method. GFF smartly decomposes computations therefore does not need to sample implicit feedback and with the proper weighting it can either be an implicit or an explicit method. (3) Both basic GFF and FM builds on SA-MDM, however the extended GFF (introduced in Section 6.5) is fully compliant with the more extensive MDM. (4) The optimization strategy of the two methods differ.

SVDFeature

Chen et. al proposed another framework, coined SVDFeature, that uses a subset of the FM model [12, 13]. Basically it assigns each attribute either to the user or to the item as a property. A feature vector is defined for each property (including the item and the user itself), and the feature vector of the item (or user) is the weighted sum of the feature vectors of its properties. The rating is predicted by the scalar product of these aggregated feature vectors. In other words, it uses a partial pairwise model that only keeps the interactions between item and user attributes. The authors claim that doing so the training time decreases drastically compared to that of FM, and the interactions dropped are mostly useless (such as interactions between metadata terms of the items). Our experiments also show that leaving out useless interactions results in more accurate models. SVDFeature can incorporate either explicit or implicit feedback as it uses a

[6] We rephrased here the feature matrix based introduction of the original paper.

[7] A certain version of ALS, which optimizes for one parameter at a time.

[8] `http://libfm.org`

ranking loss function. The model is learned using SGD. The datamodel is a dimension restricted MDM with only 2 dimensions, one for users and one for items.

The key differences between GFF and SVDFeature are as follows: (1) SVDFeature uses a fixed model, GFF takes the preference model as an input. (2) The methods use different data models, although the data model of SVDFeature is a special case of the data model of the extended GFF. (3) SVDFeature uses (pairwise) ranking loss, GFF uses pointwise ranking loss. (4) The optimization strategies differ.

Due to the incompatibility between the data model of the basic GFF and SVDFeature, and the perception of context – i.e. a context should be assigned to either the items or the users – no direct quantitative comparison is possible.

Other implicit context-aware factorization algorithms

iTALS [24] and iTALSx [23] are general factorization algorithms that use the N-way and pairwise models respectively. The key difference to GFF is that GFF does not use a fixed model. By setting the appropriate preference model, iTALS and iTALSx are special cases of GFF.

TFMAP [63] is a tensor factorization algorithm for three dimensional context-aware problems that minimizes a listwise ranking loss function with SGD on a fixed 3-way model. GFF is much more flexible as TFMAP restricts not just the model class, but also the number of dimensions. The loss function and the optimization strategy of the two methods also differ.

6.4.2 Quantitative comparison

Although I argue that the main novelty and the importance of GFF is allowing experimentation with novel models without requiring specific implementations, a quantitative comparison to Factorization Machines (state-of-the-art in context-aware factorization) and to Bayesian Personalized Ranking (state-of-the-art in handling implicit feedback) is included in this section. Both FM and BPR require the missing (negative) feedback to be sampled. I followed the steps of [45] and sampled a negative example to each positive example by replacing the item of the positive example with an item that has never occurred in the training set with the same user and context values. For FM I assigned ratings 1 and 0 to positive and negative feedbacks, respectively.

FM was trained using MCMC, as suggested by the authors of the method. The number of factors was set to $K = 80$ and the number of iterations was set to 10, because of practical requirements for the training time. Also, the method converged fairly well in 10 epochs. There were no additional hyperparameters to be optimized by FM.

The number of features and iterations was set to $K = 80$ and 10 respectively for BPR as well. The regularization coefficients and learning rate were optimized in the same way we optimized hyperparameters for GFF.

Table 6.6 shows the results (recall@20). For GFF, the pairwise, the N-way and the best non-traditional model (either interaction or context interaction model) was included. GFF outperforms FM in 3 out of 5 cases, performs very similarly in 1 case and underperforms in 1 case. GFF outperforms BPR in all cases.

W.r.t. running times, I compared FM and GFF. BPR was not included because it does not deal with context and therefore has an unfair advantage. The training time

Table 6.6: Comparison of GFF models to LibFM and BPR

Dataset	GFF N-way	 Pairwise	 Best non-traditional	LibFM	BPR
Grocery	0.1390	0.1388	**0.1669**	0.0912	0.1412
TV1	0.1315	0.1344	0.1551	**0.1683**	0.1365
TV2	0.2009	0.2323	0.3027	**0.3081**	0.1957
LastFM	0.1906	0.1873	**0.2142**	0.0652	0.2002
VoD	0.1268	0.0497	**0.1509**	0.1151	0.0539

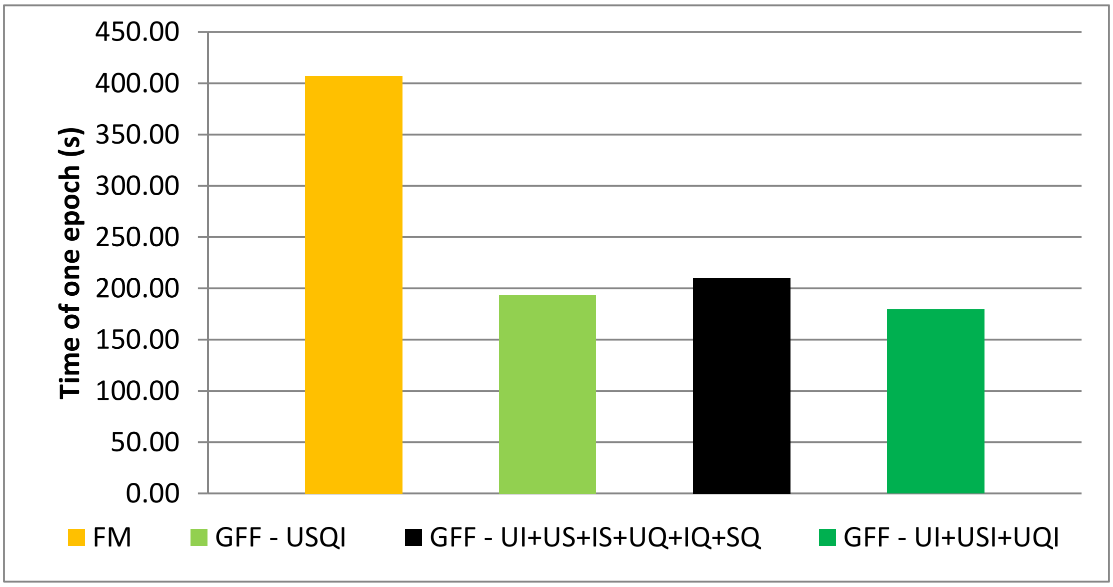

Figure 6.4: Training times of FM and GFF models on the LastFM dataset.

of FM was measured by both libFM's inner logging as well as from external code and the two values were very similar. For this measurement I did not provide a test set for libFM in order to exclude the computation of the test error. Figure 6.4 depicts the results on the LastFM dataset. GFF was twice as fast with the pairwise model and even faster with the interaction model. Due to the need of subsampling the negative feedback, FM trains on twice as many examples for the same problem. This increases the time required for training significantly. Note that the results were achieved on a single core of a multi-core CPU, and the training times of GFF can be greatly reduced if multiple cores are used in parallel.

6.5 Extension – MDM compliant GFF

In this section I release the restrictions imposed by SA-MDM and extend GFF to allow more attributes per dimensions and thus make it fully compliant with the Multidimensional Dataspace Model. More attributes per dimensions are useful for including multi-value properties of the interacting entities, e.g. tags associated with the items; session behavior; the social network of the users; etc. Such information could be also included in SA-MDM through several dimensions with a single binary attribute. Each attribute describes if a property – the value of an attribute of a dimension in MDM (see Section 1.2.1) – (e.g. a tag) applies to the entity – i.e. an instance of a dimension in MDM – (e.g. item) that participates in the transaction. The main drawback of this method is that it results in many dimensions and therefore significantly increases training times.

In my solution I intend to handle properties of entities together. This is achieved by bundling their binary attributes into one dimension in accordance with MDM. This admittedly restricts the space of possible preference models by excluding interactions between these attributes. Analogously as for context interactions, I can also argue to exclude property interactions – between properties of the same kind, since they are irrelevant from the recommendation point of view.

Our solution is inspired by NSVD1 [47] and is as follows.

1. A dimension should be defined with entities (i.e. different values of the context variable) that are associated with the properties;

2. Each property is represented by an attribute whose value denotes the strength of the attribute for a given entity. A (sparse) mixing matrix ($W \in \mathbb{R}^{S^{(P)} \times S^{(E)}}$, where $S^{(P)}$ and $S^{(E)}$ is the number of properties and entities, respectively) is formed from the values of the attributes.

3. A feature vector is assigned to each property.

4. Since each entity is the weighted sum of its properties, the feature vector of an entity is a weighted sum of the feature vectors of its properties' feature vectors. This allows the learning algorithm to be unchanged for dimensions with single attributes, because the feature vectors can be computed for the entities that directly participate in the transaction. The feature vectors of the entities can be computed using matrix multiplication: $M^{(E)} = M^{(P)}W$.

Since the derivative of the loss function w.r.t. the properties' features is not linear in the columns of $M^{(P)}$, an approximative solution is required. I chose to update the properties' feature vectors as if they were independent. To ensure convergence, after training some of the properties' feature vectors, the model should be updated before continuing. Since the update is fast, it can be done after the computation of each vector. Moreover, the update of feature vectors can be parallelized. This method can be still slow if the average number of properties assigned to entities is high.

An other way is to apply two-phase learning similarly to [49]. The first phase computes $M^{(E)}$ using a normal ALS step. In the second phase $M^{(P)}$ is computed from $M^{(E)}$ and W. The finishing step is to compute $M^{(E)} = M^{(P)}W$ from the new $M^{(P)}$, thus the following ALS steps remain consistent. Naturally, the two-phase learning is less accurate, therefore we stick to the direct optimization when possible.

Two examples are shown below on how this extension can be used.

6.5.1 Item metadata as attributes

CBF is often combined with CF to create hybrid algorithms that outperform both of them. E.g. item metadata helps overcoming the item cold-start problem in CF [10]. Here we show how to include item metadata into a model using the extended GFF.

Let us assume that the relevant item metadata is tokenized, preprocessed. From there the outline of the solution as follows.

1. We create an *item dimension*, its entities are the items, to which we will assign the metadata attributes.

2. Each metadata token is represented by an attribute that indicates the strength of a token for the items. If the item is not associated with the token, the value of the attribute is set to 0 for that item. W is created from these values.

3. A feature vector is assigned to each token.

4. The feature vectors of the items now can be computed as $M^{(I)} = M^{(M)}W$, where $M^{(M)}$ is the feature matrix of the metadata attributes.

6.5.2 Session information

Different sessions of the same user are usually treated uniformly by recommender systems, assuming that user preference does not change across sessions. Session information, however, can be of great help in identifying what the user is currently interested in. This information can further refine recommendations and is exceptionally useful in domains where users likely have broader interests (e.g. e-commerce, news sites).

As the context of the transaction, let us assign all items visited during the session but the actual one. Thus the whole session is assigned to each transaction. I excluded the actual item from the session context, since this is the prediction target. Following the outline:

1. Each transaction will be a separate entity, thus the dimension will consists of all of the transactions. The sessions can not be used as entities, because the associated attributes are different by each transaction of the session since the actual item is omitted.

2. Each item in assigned with an attribute. The attribute is either binary (i.e. the item belongs to the session or not) or weighted by the occurrences of the item in the session. W is created assigning items to each event.

3. Each item is assigned with a feature vector.

4. The feature vectors of the events now can be computed as $M^{(E)} = M^{(X)}W$, where $M^{(X)}$ is the feature matrix of the items. Note that $M^{(X)}$ is a different matrix than the feature matrix of the item dimension $M^{(I)}$.

6.5.3 Experimental evaluation

Initial experiments were done with the extended GFF to incorporate item metadata and session information into the factorization model. The general settings are the same as in Section 1.4. A user session is defined as a sequence of events of a user where the largest gap between two consecutive timestamp is less than 20 minutes. Item metadata consists of the tokenized title, description and category string of the items. The data was filtered for too common and rare tokens. For both context, the weights were ℓ_2 normalized on an entity by entity basis. The experiments were run on the Grocery dataset, because here the usage of sessions is justified and we have the necessary metadata available.

Session context is denoted by X, metadata is by M in our simplified notation. The following models were compared to the classic CF model (UI):

- $\boldsymbol{XI}$**:** Interactions between items and the session. Basically this model guesses the actual item based on the other items in the session.
- $\boldsymbol{UI+XI}$**:** The classic user–item interaction refined by the actual session.
- $\boldsymbol{UM}$**:** The items are replaced by the sum of their metadata in the classic CF model.
- $\boldsymbol{UI+UM}$**:** Two aspects of the items are used to model interaction with users, their entity and the sum of their metadata.
- $\boldsymbol{XM}$**:** Interaction between other items on the session and the metadata of the actual item.

Table 6.7: Results for the extended framework on Grocery

Model	Recall@20	Improvement
UI	0.1013	N/A
XI	0.2248	+121.97%
$UI+XI$	0.2322	+129.36%
UM	0.0614	−39.34%
$UI+UM$	0.2166	+113.87%
XM	0.2154	+112.77%

Table 6.7 summarizes the results. Note that data from the test set is needed for predicting with session in the form of other items of the test session. The results suggest that session information is very important for recommending with the Grocery dataset. XI gives strong result in and of itself and is further improved by mixing in the UI interaction as well. While metadata does not perform well in the place of items, they complements the basic UI model well and is also useful for averting the item cold-start problem.

6.6 Summary

In Chapter 6 I proposed a flexible algorithm by the name of GFF to allow for experimentation with novel preference models. The method and the results described in this chapter were published in [29] and the following theses are based on them.

Thesis 6.1 *I developed GFF (General Factorization Framework), a single, flexible factorization algorithm for the implicit feedback based context-aware recommendation problem. The flexibility of GFF lies in taking the preference model as an input. The model can use arbitrary number of dimensions and allows using any linear interaction between the subsets of aforementioned dimensions. I demonstrated that this flexibility allows for experimenting with novel preference models. The data model of the basic GFF is the single attribute MDM, which is appropriate for the context-aware problem in practice.*

Thesis 6.2 *I proposed several novel preference models for the context-aware recommendation task. I measured the usefulness of these models w.r.t. recommendation accuracy (measured by recall) on a four dimensional context-aware problem. The context dimensions I used in this problem can be generally derived from all practical datasets based on the timestamp of the events, making them especially important. I showed that there are multiple novel models that outperform traditional models used in the literature.*

Thesis 6.3 *I showed that one of the proposed models, the interaction model generally performs well. This model is the composite of the user–item (UI) and the context reweighted user–item (UCI) relations. It was the best on four datasets out of five datasets and second on the fifth one. The best model on the fifth dataset is the context interaction model that is closely related to the interaction model.*

Thesis 6.4 *I compared the recommendation accuracy of the best novel models in GFF to that of the state-of-the-art factorization methods. The novel models in GFF significantly outperformed the state-of-the-art on three out of five datasets and gave similar results on one.*

Thesis 6.5 *I extended GFF to be compliant with the Multidimensional Dataspace Model and to be able to incorporate additional information, e.g. session data and item metadata more efficiently. The preliminary experiments I executed showed that using session information can significantly increase recommendation accuracy.*

Summary

In this dissertation I focused on the implicit feedback based recommendation problem. I aimed at increasing recommendation accuracy by incorporating additional information into the recommendation. I focused on latent feature based algorithms and used mostly context information, thus the majority of this work is in the field of context-aware recommendations on implicit feedback data. The achievements are described by the following thesis groups.

Thesis Group 1: I proposed initializing matrix factorization using information on the items (or users) to increase recommendation accuracy. (See Chapter 3 for details. The methods and the results were published in [26, 27].)

> **Thesis 1.1** *I proposed to initialize the feature matrices of matrix factorization methods based on the similarities of its entities instead of starting from randomly initialized matrices. The initialization scheme is generic and thus can be applied to any matrix factorization. It consists of two steps: (1) descriptor vectors are assigned to the entities; (2) the descriptors are compressed to fit the size of the feature vectors. I applied the scheme on implicit ALS and showed on five datasets that this type of initialization can increase the recommendation accuracy measured by recall and MAP.*
>
> **Thesis 1.2** *I proposed the SimFactor algorithm that yields feature vectors, which preserve the original similarities between entities more accurately. SimFactor does not require the computation of the similarity matrix (which would be infeasible). I showed on five datasets that similarities are better estimated with this algorithm as with pure compression of the descriptor vectors. I also showed that feature vectors yielded by SimFactor are generally better for initializations than those produced by pure compression.*
>
> **Thesis 1.3** *I proposed the* Sim^2*Factor algorithm that is able to yield feature vectors whose similarity approximates the similarity between entities, based on how similar they are to the rest of the entities.* Sim^2*Factor does not require the computation of the similarity matrix. I showed that feature vectors of this kind are useful for initialization.*
>
> **Thesis 1.4** *I proposed to use context for describing entities. I showed that context based descriptors are better for initialization than metadata based ones. I also showed that the weighted combination of context and metadata based initializations can further improve the recommendation accuracy.*

Thesis Group 2: I proposed the iTALS algorithm to solve the implicit feedback based context-aware recommendation task. (See Section 4.2 of Chapter 4 for details. The method and the results were published in [24].)

> **Thesis 2.1** *I developed iTALS, a tensor factorization method that uses pointwise ranking via optimizing for weighted sum of squared errors. It estimates preferences using the N-way interaction model, i.e. the sum of elements in the elementwise product of feature vectors from each dimension. I showed that iTALS can be applied to solve the implicit feedback based context-aware recommendation problem by using ones and zeroes for positive and missing feedback respectively with higher weights for positive feedback.*
>
> **Thesis 2.2** *I showed that iTALS significantly outperforms the non context-aware implicit matrix factorization and the prefiltering based context-aware baseline with respect to recommendation accuracy, measured by recall.*
>
> **Thesis 2.3** *I demonstrated that iTALS can be trained efficiently on the implicit feedback based context-aware recommendation problem, using alternating least squares. I showed that iTALS can be efficiently used in practice as it scales linearly with the number of events and quadratically with the number of features in the range of practically useful number of feature values.*

Thesis Group 3: I proposed the iTALSx algorithm an alternative solution to the implicit feedback based context-aware recommendation task. (See Section 4.3 of Chapter 4 for details. The method and the results were published in [22, 23].)

> **Thesis 3.1** *I developed iTALSx, a tensor factorization method that uses pointwise ranking via optimizing for weighted sum of squared errors. It estimates preferences using the pairwise interaction model, i.e. the sum of dot products between feature vectors from each pair of dimensions. I showed that iTALSx can be applied to solve the implicit feedback based context-aware recommendation problem by using ones and zeroes for positive and missing feedback respectively with higher weights for positive feedback.*
>
> **Thesis 3.2** *I showed that iTALSx significantly outperforms the non context-aware implicit matrix factorization and the prefiltering based context-aware baseline with respect to recommendation accuracy, measured by recall.*
>
> **Thesis 3.3** *I demonstrated that iTALSx can be trained efficiently on the implicit feedback based context-aware recommendation problem, using alternating least squares. I showed that iTALSx can be efficiently used in practice as it scales linearly with the number of events and quadratically with the number of features in the range of practically useful number of feature values.*

Thesis Group 4: I experimented with the iTALS and iTALSx algorithms, compared them and identified easily accessible contexts. (See Section 4.4 Chapter 4 for details. The results were published in [22–24].)

> **Thesis 4.1** *I proposed to use sequentiality as context for recommendations. Sequentiality is the item with which the user previously interacted, before the current one. I argued that this context information is available with every dataset where transactions can be ordered based on their time of occurrence, which is common in practice. I showed that using this information can significantly increase recommendation accuracy to using no context and even to using seasonality as the context in a wide variety of settings (dataset, algorithms, models, number of features).*

> **Thesis 4.2** *I compared the strengths and weaknesses of iTALS (N-way model) and iTALSx (pairwise model). I found that the N-way model is more suitable when the number of features is high and/or if the dataset is denser; and the pairwise model is better otherwise.*

Thesis Group 5: I proposed ways to speed-up ALS learning through using approximate methods. (See Chapter 5 for details. The methods and the results were published in [25].)

> **Thesis 5.1** *I proposed a general, conjugate gradient based approximation for ALS in ALS based factorization algorithms. I showed that this approximation scales linearly with the number of features in the range of practically used number of feature values. I showed that this allows the usage of higher factor models and finding better trade-offs between running time and accuracy. I showed that the recommendation accuracy is affected only in a minor way if the approximation is used instead of the exact ALS.*

> **Thesis 5.2** *I proposed a general, coordinate descent based approximation for ALS in ALS based factorization algorithms. I showed that this approximation scales linearly with the number of features in the range of practically used number of feature values. I showed that this allows the usage of higher factor models and finding better trade-offs between running time and accuracy. I showed that the recommendation accuracy is affected only in a minor way if the approximation is used instead of the exact ALS.*

> **Thesis 5.3** *I compared the conjugate gradient and coordinate descent based approximate solutions from a wide variety of aspects. I showed that the conjugate gradient based method is better, because it (a) follows the exact solution more closely in terms of recommendation accuracy; (b) is faster; (c) scales better; and (d) more stable.*

> **Thesis 5.4** *I determined a good trade-off between running time and recommendation accuracy for both approximate methods. I proposed to set the number of inner iterations to 2 in order to get this trade-off.*

Thesis Group 6: I proposed a flexible algorithm by the name of GFF to allow for experimentation with novel preference models. (See Chapter 6 for details. The method and the results were published in [29].)

Thesis 6.1 *I developed GFF (General Factorization Framework), a single, flexible factorization algorithm for the implicit feedback based context-aware recommendation problem. The flexibility of GFF lies in taking the preference model as an input. The model can use arbitrary number of dimensions and allows using any linear interaction between the subsets of aforementioned dimensions. I demonstrated that this flexibility allows for experimenting with novel preference models. The data model of the basic GFF is the single attribute MDM, which is appropriate for the context-aware problem in practice.*

Thesis 6.2 *I proposed several novel preference models for the context-aware recommendation task. I measured the usefulness of these models w.r.t. recommendation accuracy (measured by recall) on a four dimensional context-aware problem. The context dimensions I used in this problem can be generally derived from all practical datasets based on the timestamp of the events, making them especially important. I showed that there are multiple novel models that outperform traditional models used in the literature.*

Thesis 6.3 *I showed that one of the proposed models, the interaction model generally performs well. This model is the composite of the user–item (UI) and the context reweighted user–item (UCI) relations. It was the best on four datasets out of five datasets and second on the fifth one. The best model on the fifth dataset is the context interaction model that is closely related to the interaction model.*

Thesis 6.4 *I compared the recommendation accuracy of the best novel models in GFF to that of the state-of-the-art factorization methods. The novel models in GFF significantly outperformed the state-of-the-art on three out of five datasets and gave similar results on one.*

Thesis 6.5 *I extended GFF to be compliant with the Multidimensional Dataspace Model and to be able to incorporate additional information, e.g. session data and item metadata more efficiently. The preliminary experiments I executed showed that using session information can significantly increase recommendation accuracy.*

Application of the results

The algorithms and the know-how resulting from this work have been successfully applied in practice. Some of the algorithms are implemented in the recommendation engine of Gravity Research & Development Inc., a recommendation service providing company with clients from all around the world in different domains. The algorithms were used successfully in the live system as well as in other recommendation projects, tenders and POCs (proof of concepts). The domains of the application include but not limited to online grocery shopping, VoD and live program recommendation on IPTV [69], e-commerce webshops and classified sites.

The results also greatly contribute to a project of the European Unions Seventh Framework Programme (FP7/2007-2013) by the name of CrowdRec[9]. CrowdRec aims for creating the next generation of (practical) recommender systems by using context, interactions with the users, analyzing streams and information from heterogenous sources. My work falls into the context related part of CrowdRec.

[9] Grant Agreement n°610594

Future research directions

There are several ways in which this work can be continued.

Most immediate research questions are related to GFF. As GFF allows for flexible experimentation, it would be interesting to have a look at other context dimension and the performance of different models with those contexts. Other meaningful context dimensions can be derived from timestamps alone (e.g. time between purchases) but some datasets have other kind of contexts (e.g. location, device) that can possibly be used for significantly improving recommendations with the proper preference model.

Although the flexibility if GFF is a great asset, finding the best preference model manually can be hard. While the interaction and context interaction (and similar) models performed well with seasonality and sequentiality and some other context dimensions, they are not necessarily the best performing models with every context. Therefore it would be useful if GFF could propose a good enough model, or in other words it would be able to learn the underlying preference model by itself. My preliminary work in this topic shows promising signs, however the proper execution of such model learning is hindered by multiple aspects of the factorization concept.

The experiments with the extended GFF are in the preliminary phase, mostly due to long training times of the method in certain cases. Although it is fast for certain data, it can be slow for others. Therefore the learning procedure could be optimized as part of future work. It would also be interesting to look more closely on the usage of MDM data, such as session information, item metadata and social networks of the users. Their interaction with items, users and context dimensions still need to be discovered.

More loosely connected research paths are related to the context itself.

One interesting question is how to assess the quality of a context dimension before using it in a model based on the data alone. This area includes (a) the assessment of context quality in a certain model; (b) the assessment of context quality in an arbitrary model (i.e. in which interactions it should be used); (c) improving context quality by combining/splitting and automatically creating context-states within a context dimension.

An other topic is the usage of non standard context dimensions in factorization. Factorization generally works with dimensions that contain atomic entities but no relation is assumed between said entities. However in some contexts the entities are ordered or non atomic or there are multiple levels of the context (hierarchical context) or the context dimension is continuous. Classically all of these context dimensions are transformed in a way to be conform with the needs of factorization, but this results in information loss. My preliminary research in this area [28] demonstrates that better modeling of these dimensions results in increased recommendation accuracy. However my work in this area only scratched the surface and there is still a lot to be discovered.

Finally, the wider area of my work is integrating other information into latent feature based recommendation algorithms be it through using context, item metadata, session information, etc. These information sources provide well structured data that does not require huge amounts of preprocessing to use. However it would be more interesting to include other information in the recommendation algorithms, such as images (of products), low level audio features for music recommendation, video frames for video recommendation or unstructured textual data (e.g. product descriptions). With the recent advancements in the field of neural networks and deep learning this seems to be possible to do in the next few years (at least in the lab). My preliminary research in this area shows great potential.

Bibliography

[1] G. Adomavicius and F. Ricci. Workshop on context-aware recommender systems (CARS-2009). In *Recsys'09: ACM Conf. on Recommender Systems*, pages 423–424, 2009. ISBN 978-1-60558-435-5.

[2] G. Adomavicius and A. Tuzhilin. Context-aware recommender systems. In *Recsys'08: ACM Conf. on Recommender Systems*, pages 335–336, 2008.

[3] G. Adomavicius, R. Sankaranarayanan, S. Sen, and A. Tuzhilin. Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Trans. Inf. Syst.*, 23(1):103–145, 2005. ISSN 1046-8188.

[4] R. Bader, E. Neufeld, W. Woerndl, and V. Prinz. Context-aware POI recommendations in an automotive scenario using multi-criteria decision making methods. In *CaRR'11: Workshop on Context-awareness in Retrieval and Recommendation*, pages 23–30, 2011. ISBN 978-1-4503-0625-6.

[5] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[6] Márton Balassi, Robert Pálovics, and András A. Benczúr. Distributed frameworks for alternating least squares. In *2nd Large Scale Recommender Systems Workshop at Recsys 2014*, Foster City, CA, USA, 2014.

[7] R. Bell and Y. Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *ICDM'07: IEEE Int. Conf. on Data Mining*, pages 43–52, 2007.

[8] J. Bennett and S. Lanning. The Netflix Prize. In *KDD Cup Workshop at SIGKDD'07*, pages 3–6, 2007.

[9] Derek Bridge, Mehmet H Göker, Lorraine McGinty, and Barry Smyth. Case-based recommender systems. *The Knowledge Engineering Review*, 20(03):315–320, 2005.

[10] Robin Burke. Hybrid web recommender systems. In *The adaptive web*, pages 377–408. Springer, 2007.

[11] O. Celma. *Music Recommendation and Discovery in the Long Tail*. Springer, 2010.

[12] T. Chen, Z. Zheng, Q. Lu, W. Zhang, and Y. Yu. Feature-based matrix factorization. *CoRR*, abs/1109.2271, 2011.

[13] T. Chen, W. Zhang, Q. Lu, K. Chen, Z. Zheng, and Y. Yu. SVDFeature: A toolkit for feature-based collaborative filtering. *Journal of Machine Learning Research*, 13: 3619–3622, 2012.

[14] P. Cremonesi and R. Turrin. Analysis of cold-start recommendations in IPTV systems. In *Recsys'09: ACM Conf. on Recommender Systems*, 2009.

[15] Mukund Deshpande and George Karypis. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems (TOIS)*, 22(1):143–177, 2004.

[16] Ricardo Dias and Manuel J Fonseca. Improving music recommendation in session-based collaborative filtering by using temporal context. In *Tools with Artificial Intelligence (ICTAI), 2013 IEEE 25th International Conference on*, pages 783–788. IEEE, 2013.

[17] Zeno Gantner, Steffen Rendle, and Lars Schmidt-Thieme. Factorization models for context-/time-aware movie recommendations. In *Proceedings of the Workshop on Context-Aware Movie Recommendation*, CAMRa '10, pages 14–19, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0258-6. doi: 10.1145/1869652.1869654. URL `http://doi.acm.org/10.1145/1869652.1869654`.

[18] Jennifer Golbeck. *Generating predictive movie recommendations from trust in social networks*. Springer, 2006.

[19] GroupLens Research. Movielens data sets, 2006. `http://www.grouplens.org/node/73`.

[20] Ido Guy, Naama Zwerdling, David Carmel, Inbal Ronen, Erel Uziel, Sivan Yogev, and Shila Ofek-Koifman. Personalized recommendation of social software items based on social relations. In *Proceedings of the third ACM conference on Recommender systems*, pages 53–60. ACM, 2009.

[21] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, pages 409–436, 1952.

[22] B. Hidasi. Technical report on iTALSx. Tech. Report Series 2012-2, Gravity R&D Inc., 2012.

[23] B. Hidasi. Factorization models for context-aware recommendations. *Infocommunications Journal*, VI(4):27–34, 2014.

[24] B. Hidasi and D. Tikk. Fast ALS-based tensor factorization for context-aware recommendation from implicit feedback. In *ECML-PKDD'12, Part II*, number 7524 in LNCS, pages 67–82. Springer, 2012.

[25] B. Hidasi and D. Tikk. Speeding up ALS learning via approximate methods for context-aware recommendations. *Knowledge and Information Systems*, pages 1–25, 2015. ISSN 0219-1377. doi: 10.1007/s10115-015-0863-2. URL `http://dx.doi.org/10.1007/s10115-015-0863-2`.

[26] Balázs Hidasi and Domonkos Tikk. Enhancing matrix factorization through initialization for implicit feedback databases. In *Proceedings of the 2nd Workshop on Context-awareness in Retrieval and Recommendation*, pages 2–9. ACM, 2012.

[27] Balázs Hidasi and Domonkos Tikk. Initializing matrix factorization methods on implicit feedback databases. *J. UCS*, 19(12):1834–1853, 2013.

[28] Balázs Hidasi and Domonkos Tikk. Approximate modeling of continuous context in factorization algorithms. In *Proceedings of the 4th Workshop on Context-Awareness in Retrieval and Recommendation*, CARR '14, pages 3–9, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2723-7. doi: 10.1145/2601301.2601303. URL `http://doi.acm.org/10.1145/2601301.2601303`.

[29] Balázs Hidasi and Domonkos Tikk. General factorization framework for context-aware recommendations. *Data Mining and Knowledge Discovery*, pages 1–30, 2015. ISSN 1384-5810. doi: 10.1007/s10618-015-0417-y. URL `http://dx.doi.org/10.1007/s10618-015-0417-y`.

[30] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM'08: IEEE Int. Conf. on Data Mining*, pages 263–272, 2008.

[31] M. Jahrer and A. Töscher. Collaborative filtering ensemble for ranking. In *KDD Cup Workshop at 17th ACM SIGKDD'11*, 2011.

[32] I.T. Jolliffe. *Principal Component Analysis.* Springer Verlag, 1986.

[33] A. Karatzoglou, X. Amatriain, L. Baltrunas, and N. Oliver. Multiverse recommendation: N-dimensional tensor factorization for context-aware collaborative filtering. In *Recsys'10: ACM Conf. on Recommender Systems*, pages 79–86, 2010. ISBN 978-1-60558-906-0.

[34] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *SIGKDD'08: ACM Int. Conf. on Knowledge Discovery and Data Mining*, pages 426–434, 2008.

[35] Mark A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):233–243, 1991. ISSN 1547-5905. doi: 10.1002/aic.690370209. URL `http://dx.doi.org/10.1002/aic.690370209`.

[36] L. Lathauwer, B. Moor, and J. Vandewalle. A multilinear singular value decomposition. *SIAM J. Matrix Anal. Appl.*, 21(4):1253–1278, 2000. ISSN 0895-4798.

[37] G. Linden, B. Smith, and J. York. Amazon. com recommendations: Item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, 2003.

[38] Roderick J. A. Little and Donald B. Rubin. *Statistical Analysis with Missing Data.* John Willey & Sons, Inc., 1987.

[39] N. N. Liu, B. Caoand M. Zhao, and Q. Yang. Adapting neighborhood and matrix factorization models for context aware recommendation. In *CAMRa'10: Workshop on Context-Aware Movie Recommendation*, pages 7–13, 2010. ISBN 978-1-4503-0258-6.

[40] Qiwen Liu, Tianjian Chen, Jing Cai, and Dianhai Yu. Enlister: Baidu's recommender system for the biggest Chinese Q&A website. In *RecSys-12: Proc. of the 6th ACM Conf. on Recommender Systems*, pages 285–288, 2012.

[41] Andreas Lommatzsch. Real-time news recommendation using context-aware ensembles. In Maarten de Rijke, Tom Kenter, ArjenP. de Vries, ChengXiang Zhai, Franciska de Jong, Kira Radinsky, and Katja Hofmann, editors, *Advances in Information Retrieval*, volume 8416 of *Lecture Notes in Computer Science*, pages 51–62. Springer International Publishing, 2014. ISBN 978-3-319-06027-9. doi: 10.1007/978-3-319-06028-6_5.

[42] P. Lops, M. Gemmis, and G. Semeraro. Content-based recommender systems: State of the art and trends. In *Recommender Systems Handbook*, pages 73–105. Springer, 2011.

[43] Tariq Mahmood and Francesco Ricci. Towards learning user-adaptive state models in a conversational recommender system. In *LWA*, pages 373–378. Citeseer, 2007.

[44] Paolo Massa and Paolo Avesani. Trust-aware collaborative filtering for recommender systems. In *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, pages 492–508. Springer, 2004.

[45] Trung V. Nguyen, Alexandros Karatzoglou, and Linas Baltrunas. Gaussian process factorization machines for context-aware recommendations. In *SIGIR-14: ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 63–72, 2014.

[46] Douglas W Oard, Jinmook Kim, et al. Implicit feedback for recommender systems. In *Proceedings of the AAAI workshop on recommender systems*, pages 81–83, 1998.

[47] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proc. of KDD Cup and Workshop*, pages 5–8, 2007.

[48] I. Pilászy. *Factorization-Based Large Scale Recommendation Algorithms.* PhD thesis, Budapest University of Technology and Economics, 2010.

[49] I. Pilászy and D. Tikk. Recommending new movies: Even a few ratings are more valuable than metadata. In *Recsys'09: ACM Conf. on Recommender Systems*, pages 93–100, 2009. ISBN 978-1-60558-435-5.

[50] I. Pilászy, D. Zibriczky, and D. Tikk. Fast ALS-based matrix factorization for explicit and implicit feedback datasets. In *Recsys'10: ACM Conf. on Recommender Systems*, pages 71–78, 2010. ISBN 978-1-60558-906-0.

[51] S. Rendle. Factorization machines with libFM. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(3):57, 2012.

[52] S. Rendle and L. Schmidt-Thieme. Pairwise interaction tensor factorization for personalized tag recommendation. In *WSDM'10: ACM Int. Conf. on Web Search and Data Mining*, pages 81–90, 2010. ISBN 978-1-60558-889-6.

[53] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. BPR: Bayesian personalized ranking from implicit feedback. In *UAI'09: 25th Conf. on Uncertainty in Artificial Intelligence*, pages 452–461, 2009. ISBN 978-0-9749039-5-8.

[54] S. Rendle, Z. Gantner, C. Freudenthaler, and L. Schmidt-Thieme. Fast context-aware recommendations with factorization machines. In *SIGIR'11: ACM Int. Conf. on Research and Development in Information*, pages 635–644, 2011. ISBN 978-1-4503-0757-4.

[55] Steffen Rendle. Scaling factorization machines to relational data. In *PVLDB'13: 39th Int. Conf. on Very Large Data Bases*, pages 337–348, 2013.

[56] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186. ACM, 1994.

[57] Francesco Ricci, Dario Cavada, Nader Mirzadeh, and Adriano Venturini. Case-based travel recommendations. *Destination Recommendation Systems: Behavioural Foundations and Applications*, pages 67–93, 2006.

[58] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. *Recommender Systems Handbook*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010. ISBN 0387858199, 9780387858197.

[59] A. Said, S. Berkovsky, and E. W. De Luca. Putting things in context: Challenge on context-aware movie recommendation. In *CAMRa'10: Workshop on Context-Aware Movie Recommendation*, pages 2–6, 2010. ISBN 978-1-4503-0258-6.

[60] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems 20*. MIT Press, 2008.

[61] J Ben Schafer, Joseph A Konstan, and John Riedl. E-commerce recommendation applications. In *Applications of Data Mining to Electronic Commerce*, pages 115–153. Springer, 2001.

[62] Andrew I Schein, Alexandrin Popescul, Lyle H Ungar, and David M Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 253–260. ACM, 2002.

[63] Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, Alan Hanjalic, and Nuria Oliver. TFMAP: Optimizing MAP for top-N context-aware recommendation. In *SIGIR'12: ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 155–164, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1472-5. doi: 10.1145/2348283.2348308.

[64] Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, Nuria Oliver, and Alan Hanjalic. Climf: Learning to maximize reciprocal rank with collaborative less-is-more filtering. In *Proceedings of the Sixth ACM Conference on Recommender*

Systems, RecSys '12, pages 139–146, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1270-7. doi: 10.1145/2365952.2365981. URL `http://doi.acm.org/10.1145/2365952.2365981`.

[65] G. Takács and D. Tikk. Alternating least squares for personalized ranking. In *Recsys'12: 6th ACM Conf. on Recommender Systems*, pages 83–90, 2012.

[66] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Major components of the Gravity recommendation system. *SIGKDD Explor. Newsl.*, 9:80–83, December 2007. ISSN 1931-0145.

[67] G. Takács, I. Pilászy, and D. Tikk. Applications of the conjugate gradient method for implicit feedback collaborative filtering. In *RecSys'11: ACM Conf. on Recommender Systems*, pages 297–300, 2011.

[68] Raafat Zarka, Amélie Cordier, Elöd Egyed-Zsigmond, and Alain Mille. Contextual trace-based video recommendations. In *Proceedings of the 21st International Conference Companion on World Wide Web*, WWW '12 Companion, pages 751–754, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1230-1. doi: 10.1145/2187980.2188196. URL `http://doi.acm.org/10.1145/2187980.2188196`.

[69] Dávid Zibriczky, Balázs Hidasi, Zoltán Petres, and Domonkos Tikk. Personalized recommendation of linear content on interactive tv platforms: beating the cold start and noisy implicit user feedback. In *UMAP Workshops*, 2012.