LOW RANK MODELS FOR RECOMMENDER SYSTEMS WITH LIMITED PREFERENCE INFORMATION

Doctoral Thesis

by

EVGENY FROLOV

DOCTORAL PROGRAM IN COMPUTATIONAL AND DATA INTENSIVE SCIENCE AND ENGINEERING

Supervisor
Professor Ivan Oseledets

Moscow – 2018

Contents

List of Figures

List of Tables

Listings

Abstract

Learning from a collective behavior of crowds to predict individual user preferences is one of the major tasks in recommender systems. Among the key challenges that make this task especially difficult is the very fact that behavioral data is inherently incomplete, which leaves the room for various assumptions. One of the core assumptions implicitly used by conventional collaborative filtering models is that, despite a largely missing data, it is still possible to uncover reliable patterns for generating relevant recommendations. This, however, is not always the case and the resulting models in practice tend to exhibit poor performance when the fraction of known user preferences decreases.

In the light of this problem, we examine the shortcomings of a particular SVD-based model, namely PureSVD, which in many cases outperforms other state-of-the-art approaches and, most importantly, provides a number of practical advantages over other solutions. In order to address its limitations, related to the lack of preference data, without sacrificing its key benefits, we revisit the problem of a low rank approximation and derive several generalized formulations based on classical results from linear and multilinear algebra. As a result, we propose three new methods that use both matrix and tensor factorization techniques.

All proposed methods tackle the problem of insufficient preference information from different angles. The first tensor-based approach improves the warm start regime. The hybrid SVD-based approach is suitable for extremely sparse data, which is a frequent problem in many domains. It also partially addresses the cold start problem. As a combination of these two methods, the third method inherits the key advantages of both predecessors and at the same time allows to compensate for their major pitfalls: an increased susceptibility to a high degree of data sparsity in the tensor case and an emergence of undesired spurious correlations in the matrix case.

We evaluate our models on several benchmark datasets commonly used by researchers in recommender systems field. Based on experimental results we justify the choice of each particular model depending on the usage scenario and the properties of input data. All experiments are performed with the help a new open-source recommendation framework named Polara, which is developed by the author of the thesis to facilitate an in-depth quality evaluation, support quick model prototyping and to ensure research reproducibility.

Publications

1. “Fifty shades of ratings: How to Benefit from Negative Feedback in Top-n Recommendations Tasks”, Evgeny Frolov and Ivan Oseledets; Proceedings of the 10th ACM Conference on Recommender Systems, 2016, pp. 91-98.

2. “Tensor methods and recommender systems”, Evgeny Frolov and Ivan Oseledets; WIREs Data Mining Knowledge Discovery 2017, vol. 7, issue 3.

3. “Matrix Factorization in collaborative filtering”, Evgeny Frolov and Ivan Oseledets; Collaborative Recommendations: Algorithms, Practical Challenges and Applications, book chapter, to be published by World Scientific Publishing Co. Pte. Ltd. in Summer 2018.

4. “HybridSVD: When collaborative Information is Not Enough”, Evgeny Frolov and Ivan Oseledets; arXiv:1802.06398, 2017.

5. “Revealing the Unobserved by Linking Collaborative Behavior and Side Knowledge”, Evgeny Frolov and Ivan Oseledets; arXiv:1807.10634, 2018.

Thesis outline

Significance. The thesis is devoted to the development of new low rank models for recommender systems. The work focuses on matrix- and tensor-based factorization techniques [94, 54] widely used in industry. Matrices and tensors naturally arise in the collaborative filtering approach, where information about collective human behavior is used to build a recommendation model. In these settings, low rank methods allow to conveniently model interactions between users and items and compactly represent them in terms of a small number of *latent features*. This allows providing scalable solutions, capable of dealing with millions of users and items which is a common requirement in modern systems. Another important aspect is the ability to work in highly dynamic online environments where users expect an immediate response from a system to their actions. Many factorization techniques help to address that problem by the means of the so-called *folding-in* approach [49].

Although various factorization techniques have been already developed to date, there are still certain scenarios where a simple well known SVD-based model called *PureSVD* [38] and its derivatives [112] allow to outperform other state-of-the-art approaches. It provides many additional benefits, such as lower storage requirements and simplified model fine-tuning; it is based on a stable algorithm with deterministic output and gives an efficient analytic solution for folding-in, which makes it especially attractive for practical applications. Moreover, the SVD algorithm has highly optimized implementations in many programming languages. However, it also has certain limitations and drawbacks.

First of all, as any standard matrix factorization-based model, PureSVD is sensitive to input data. More specifically, the value of user feedback (e.g., rating) affects the ability of the model to learn user interests: lower ratings will have a lower contribution into the result, and the feedback with higher rating values will dominate.

However, low ratings may provide as a strong signal about actual user preferences as high ratings and, therefore, should not be neglected. The inability to take that into account potentially leads to an increased rate of irrelevant recommendations. This may play a crucial role, especially in the case when a new user has only started to interact with the system and has not provided a sufficient amount of information about his or her preferences yet. Too many irrelevant recommendations, in that case, lower the credibility of the service and user may never want to use it again.

Secondly, PureSVD relies on collaborative information only and ignores any side information such as user attributes or item features. Given that interaction data is often very scarce, this may prevent the model from reliably learning intrinsic relations within the data and lead to considerable degradation of recommendations' quality. In such cases side information may serve as a valuable source of additional relational data and may help to reveal important patterns, making the model more resistant to the lack of collaborative information. In addition to that, side information allows to alleviate the *cold start* problem when interactions for a user or an item data are not yet available. However, the question of generalizing the purely SVD-based approach to include both collaborative and side information is still an open research problem.

The primary goal of the thesis work is to develop efficient low rank factorization methods, which inherit the key benefits of the PureSVD approach and do not suffer from its major limitations induced by the lack of known user preferences.

Novelty. Three new methods based on matrix and tensor factorizations are proposed in this work. The first method treats user feedback as an ordinal concept in contrast to the commonly used real-valued representation. In this method, (*user, item, feedback*) triplets are encoded into a third order tensor which is factorized with the help of the Tucker decomposition. This allows to preserve orthogonality of factors and, similarly to the PureSVD approach, leads to an analytic solution for folding-in. More importantly, the model becomes equally sensitive to any user feedback and allows to explicitly impose ordinal relations within the data which helps to generate relevant recommendations even from negative-only feedback. This is especially important in the warm start, settings when only a little information about user preferences is known. In these settings, the model performs much better in terms of avoid-

ing irrelevant recommendations and may help users find relevant content much faster. Due to common positivity bias (the tendency to favor highly rated items) of both conventional recommendation models and standard evaluation metrics, a new debiased evaluation methodology is also proposed. It provides a more detailed view on the quality of recommendations, which takes into account not only a user satisfaction but also a potential user disappointment.

The second proposed method extends PureSVD model with side information. While there exists a number of factorization techniques, which allow taking side information into account (the so-called *hybrid approach*), they use different optimization algorithms that do not provide the same set of benefits as SVD. In contrast, the proposed approach allows to incorporate user attributes and item features directly into the model while staying within the SVD-based computation paradigm. This is achieved with the help of the generalized SVD formulation. An efficient computational scheme involving Cholesky decomposition is proposed to make the model suitable for large scale data. The method inherits all the advantages of the original PureSVD approach, including an analytical form of folding-in, and resolves the problem of insufficient preference data.

The third proposed method combines formulations of the previous two methods into a single higher order factorization model. It incorporates the hybrid part of the SVD-based approach into the Tucker decomposition. The corresponding new and efficient optimization technique, which takes the specific structure of the problem into account, is provided. The combined approach inherits the key advantages of both predecessors and at the same time allows to compensate for their major pitfalls: an increased susceptibility to a high degree of data sparsity in the tensor case and an emergence of undesired spurious correlations in the matrix case.

Practicality. Any recommendation system faces the limited preferences information problem. Both cold start and warm start cases are the most frequent examples of that. More globally, even information about known users and items in an established system is sometimes insufficient given that the variety of choice in a digital world may grow far beyond human capabilities of exploring it. The problem may cause the so-called “trust busters” effect when too many irrelevant recommendations make users feel the service is not good enough. Disappointed users may stop using the service and

never want to return to it again. The proposed approaches tackle this problem from different angles. The tensor-based approach focuses on avoiding irrelevant recommendations by treating both positive and negative user feedback more appropriately. The hybrid SVD-based approach employs side information and attempts to recover possible unobserved relations even if no preference data is available at all. This also allows to work with extremely sparse data, which is a frequent problem in the domains with large item assortments (e.g., online retail stores). As a combination of these two methods, the third tensor-based hybrid approach takes the best of their properties. It aims to simultaneously increase the perceived quality of a recommendation service, maintain high user satisfaction and ensure high user retention and user loyalty in the long run.

Structure. The thesis is divided into *three major parts* consisting of one or several chapters. The *first part* is introductory. It starts with a chapter on some general concepts in the recommender systems field, followed by two chapters that provide an overview of common matrix- and tensor-based factorization methods respectively. The fourth chapter gives the necessary background on the main topic of this work and presents a view on the current state of the field in this respect. The *second part* includes three chapters that provide a detailed description of all three newly proposed methods respectively. The *third part* focuses on software aspects and introduces a new open-source recommendation framework Polara, developed by the author of the thesis as a part of his research.

Acknowledgements

Writing a thesis work is a significant commitment that requires a lot of efforts, time and energy. This is also a memorable journey, which wouldn't be possible without all the great people that were around me all these years. I would like to express my gratefulness to all of them for making it a unique and delightful experience.

Of course, nothing of this would be ever possible without my supervisor – Ivan Oseledets – who not only gave me an opportunity to start a new page in my professional life but also provided me with exceptional guidance and comprehensive support. A bit more than four years ago it was his distinctive teaching style and inspirational way of expressing scientific ideas that motivated me to join his brilliant group. Since then there was not a moment that I would regret my decision.

I would like to express words of appreciation to Maxim Rakhuba, who also played a crucial role in my professional development. His positive influence on my understanding of what it means to be a young scientist is hard to overestimate. Maxim was helping me to catch up with many relevant topics, crucial for my research, and conversations with him were always very productive, insightful and pleasing. His patience and willingness to help with non-trivial ideas, a never-ending encouragement and optimism were invaluable to me.

I also feel grateful to Michael Thess for a gentle and very informative introduction to the field of recommender systems. Now, by looking back, I can clearly say that it would be much harder to start off without all the insights and experience that Michael kindly shared with me. I appreciate the opportunity to work with Michael and his research team in the prudSys company during my two internships there.

Part I

Overview of low rank models in recommender systems

Chapter 1

General concepts

We live in the era of data explosion and information overload. Managing it would be impossible without the help of intelligent systems that can process and filter huge amounts of information much faster than humans. The need for such systems was already recognized in late 1970s in the Usenet, a distributed discussion platform, founded at Duke University. One of its goals was to help users to maintain numerous posts by grouping them into newsgroups. However, an active research on the topic of information filtering started in 1990s. As it was noted in [4], the general term Recommender Systems (RS) was brought to the academia in the mid-90's with works of Resnick, Hill, Shardanand and Maes and, according to [25], was preceded by several famous projects: Tapestry, Lotus Notes, GroupLens. A significant boost in RS research started after a famous Netflix prize competition with $1 million award for the winners, announced back in 2006. This has not only attracted a lot of attention from scientists and engineers, but also depicted the great interest from industry.

1.1 Recommender systems at a glance

Let us consider without loss of generality the task of product recommendations. The main goal of this task is, given some prior information about users and items (products), try to predict what particular items will be the most relevant to a selected user. The relevance is measured with some relevance score (or utility) function f_u and can be schematically described as

$$f_u : \text{User} \times \text{Item} \to \text{Relevance Score}, \tag{1.1}$$

where User is a domain of all users and Item is a domain of all items.

There are several ways how this utility can be estimated from real data: either based on the observed user feedback or based on available characteristics of users and items. Feedback data can be either explicit or implicit, depending on whether it is directly provided in the form of explicit user preferences (e.g. ratings, likes/dislikes) or implicitly collected through an observation of user actions (e.g. page clicks, product purchases). In turn, characteristics may consist of various intrinsic features and attributes. For example, users can be described by an age, gender or other demographic data, while items may belong to some category and be additionally characterized by a list of key properties.

Availability of the described two types of prior information defines what class of techniques will be used to generate recommendations. When only preference data can be accessed, then this is a task for the so called *collaborative filtering* (CF) approach. Alternatively, if only item properties and user attributes are available, then the recommendation problem is solved with the *content-based* (CB) approach.

1.1.1 Content-based filtering

The CB approach exploits knowledge about user attributes and item features in order to find the best matching (*user, item*) pairs that can be used to generate the most relevant recommendations. This approach relies on the assumption that user choice is influenced by a combination of certain item properties and individual user features. As a trivial example, users with a certain income may prefer products of a particular category or brand. In practice, such relations may have much more complicated nature and require good domain knowledge in order to take into account more intricate cases.

One of the main advantages of the CB approach is the ability to alleviate the so called *cold start* problem (Sec. 1.2.1), where preference data is unavailable and CF algorithms are simply inapplicable. As long as all the needed content information is collected, recommendations can be produced instantly even in the case of items that were never recommended to any user before.

The focus on content features, however, leads to a number of issues, such as limited content analysis, over-specialization and high sensitivity to user input [4, 102], which decrease the perceived quality of recommendations. Beyond that, from purely practical point of view, it can be quite challenging to get descriptive and reliable

data, as users are not always motivated enough to provide comprehensive information about themselves and items may have incomplete or corrupted descriptions.

More importantly, users' *decision making process typically has a multifaceted nature* and can be influenced by various internal and external aspects, which may not necessarily be available for observation and may not align well with the collected characteristic data. Therefore, the use of CB methods alone can be quite limiting. On the other hand, while users interact with RS, they leave a "trace" of how actual decisions are made. This information, if properly collected, can be used to uncover some common patterns in user behavior that could potentially help to generate more reasonable and relevant recommendations. This leads to the CF approach.

1.1.2 Collaborative filtering

In contrast to CB filtering, the CF approach does not require any specific knowledge about users or items and only uses prior observation of collective user behavior in order to generate new recommendations. It helps uncover general patterns from collective behavior, even if it is governed by a set of unidentifiable effects, events, motives, etc. All CF methods are generally divided into two categories: *memory-based* and *model-based* techniques. This classification was initially proposed in [24] and became standard in the field [21, 2].

Memory-based collaborative filtering

A widely used and very popular approach in this category is based on the k *Nearest Neighbours* (kNN) algorithm [73]. It finds relevance scores for any (*user, item*) pair by considering weighted contributions of its neighbors. The neighborhood is typically determined by a similarity between either users (user-based approach) or items (item-based approach) [148] in terms of some pre-defined similarity measure. This is also called a similarity-based approach. In its simplest implementation, the method requires to store in memory all similarity coefficients as well as prior information about user-item interactions in order to make predictions.

Performance of the similarity-based models can be greatly impacted by a selected measure of similarity (or a distance measure). Cosine similarity, Jaccard index, Pearson correlation, Okapi BM25 [120] are a few examples of possible choices. Even

though such models may give a good recommendation quality in some application domains, factorization models (see next section) are better suited for large-scale problems often met in practice, providing high computational performance and delivering high quality recommendations [92, 21].

Model-based collaborative filtering

In the model-based approach a long enough history of observations is used to build a predictive model first. Such models use collective behavior of a crowd (a "wisdom of crowds") in order to extract general behavioral patterns and represent it in a convenient, typically compact, form. Among the most successful model-based approaches are matrix factorization (MF), described in details in Chap. 2, and higher-order tensor factorization (TF), described in Chap. 3.

The power of factorization models comes from the ability to embed users, items and other entities involved in interactions between the former two as vectors in a lower dimensional space of latent (also called hidden) features. These vectors not only allow to describe the observed consumption patterns, but also help to predict previously unseen user preferences and find new relevant items. For example, in the matrix case the relevance score that corresponds to any user-item interaction can be simply measured as an inner product between their vectors in the latent feature space.

1.1.3 Hybrid recommenders

Both CF and CB approaches tackle the problem of generating relevant recommendations from very different angles and have their own set of advantages and disadvantages. Many successful RS use *hybrid* approach [26] to accumulate strengths of both methods within a single model and compensate for their disadvantages. This allows to improve recommendations not only in standard cases, but also in such extreme scenarios as cold start (Sec. 1.2.1).

Hybrid recommender systems are closely related to the field of ensemble analysis in standard classification tasks. For example, you can treat collaborative filtering models as a generalization of classification models. All ensemble systems in that respect are hybrid models. The opposite, however, is not necessarily true. There are 3 top-level design patterns for building hybrid recommender systems.

The first one is *ensemble*. In this setup pre-selected recommender systems are used in the true "black-box" or "off-the-shelf" fashion. Every model in the ensemble produces scores in a unified way so that all the models are interchangeable and their outputs can be easily combined. As an example, one could combine predictions of the latent factors and the neighborhood-based models.

The second type is *monolithic* systems that are designed for heterogenous setups with different sources of data or different classes of recommender models fused together. One cannot use them in a purely "black-box" mode, as they typically require additional efforts to process the input data and merge different algorithmic parts. In some cases it may not be even possible to have a clear distinction between these parts.

Finally, *mixed* systems simultaneously present the outputs of several recommender models. As an example, consider an online shop with several blocks of recommendations displayed in different locations of the web page. One block may be responsible for current shopping trends, while the other one for more tailored recommendations, based on a visitor's purchase history.

This high-level taxonomy can be further divided into a number of more specific representative classes. Description of all the classes goes beyond the scope of this work. However, we note one particular case of monolithic design based on the *feature combination* pattern, where several heterogeneous data sources are combined and then are used within a unified recommendation model. This pattern can be found in a variety of hybrid recommenders, including some factorization methods, and will be also used in this work (see Chap. 6 and Chap. 7).

1.2 Challenges for recommender systems

Building high quality RS is a complex problem that involves not only a certain level of scientific knowledge, but also greatly depends on practical experience, accumulated in real world applications. This makes the topic of challenges very broad and we will only briefly discuss some of the most common aspects closely related to initial model design and algorithmic implementations.

1.2.1 Cold start

Cold start is the problem of handling new entities, i.e. users or items or, in the most sever case of the system cold start – both at the same time [49]. For example, when a new (or unrecognized) user is introduced to the system, information about user preferences is yet unavailable, which makes it nearly impossible to predict any interesting and relevant items. Similar problem arises when a new item appears in a product catalog. If an item has no content description or it was not rated by any user it will be impossible to build recommendations with this item.

Even after a few interactions, i.e. in the so called *warm start* case, predicting preferences can still be quite a difficult task prone to unintended biases. This, however, gives more space for maneuver. Unlike the cold start case, where CF models are simply inapplicable, knowing a few preferences allows to employ some sort of incremental approach (e.g., folding-in, see Sec. 2.2.3) in order to quickly update a CF model with new information and generate new recommendations.

1.2.2 Missing values

Users naturally engage with only a small subset of items and a considerable amount of possible interactions stays unobserved. Collecting more data requires time and efforts and its availability depends on various factors related to the domain of application and user habits, which makes the task quite difficult. Excluding the trivial case of the lack of interest in specific items, there can be many other reasons why users do not interact with them. For instance, users can simply be unaware of existing alternatives for the items of their choice or just face interesting items not at the right moment. Finding out such reasons helps to make better predictions. This task, however, is accompanied with a high level of uncertainty, which may bring an undesirable bias against unobserved data or even prevent recommender models from learning representative patterns.

This is especially critical for the CF approach as it relies on the assumption that collaborative information is sufficient, i.e. accommodates all important variations in user behavior, so that intrinsic relations can be reliably learned solely from that data. However, there is some evidence that *when the observed user-item interactions are too*

scarce, CF models may fail to generalize well and tend to produce unreliable predictions [198, 5].

There are several commonly used techniques that help to alleviate these issues and improve RS quality. In the MF case, simple regularization may prevent undesired biases. Another effective technique is to assign some non-zero weights to the missing data, instead of completely ignoring it [82]. Hybrid models can take advantage of content information in order to pre-process observations and assign non-zero relevance scores to some of the unobserved interactions, which is sometimes called *sparsity smoothing*. Alternatively, content information can be used to add additional regularization. Data clustering is another effective approach, which is typically used to split the problem into a number of subproblems of a smaller size with a more connected information within each cluster.

Nevertheless, when data sparsity is not too extreme, even simpler methods can work quite well. In the case of MF methods based on Singular Value Decomposition (SVD) [65], simply imputing zeros in place of unobserved values is sometimes sufficient [38, 96]. Additional smoothing can be achieved in that case with the help of a *kernel trick* [154]. Other missing value imputation techniques based on various data averaging and normalization methods are also possible [49].

1.2.3 Implicit feedback

In many real systems users are not motivated or not technically equipped to provide any information about their actual experience after interacting with an item. Hence, user preferences can only be inferred from an implicit feedback, which may not necessarily reflect the actual user taste or even tell with guarantees whether the user likes an item or dislikes it [82].

1.2.4 Model evaluation

Without a well designed evaluation workflow and an adequate quality measure it is impossible to build a reliable RS model that behaves equally well in both laboratory and production environments. Moreover, there are many aspects of a model assessment beyond recommendation accuracy that are related to both user experience and business goals. This includes metrics like *coverage, diversity, novelty, serendipity*

[153], and indicators such as total generated revenue or average revenue per user session. This is still an open and ongoing research problem as it is not totally clear what are the most relevant and informative offline metrics and how to align them with the real online performance.

The most reliable evaluation of RS performance is an online testing and user studies. However, researchers typically do not have an access to production systems so a number of offline metrics, mostly borrowed from IR field, became widely used as an alternative. The most important among them are the relevance metrics: precision, recall, F1-score; and the ranking metrics: normalized discounted cumulative gain (NDCG), mean average precision (MAP), mean reciprocal rank (MRR), area under the ROC curve (AUC). These metrics may to some extent simulate a real environment, and in same cases have a direct correlation with business metrics (e.g., recall and clickthrough rates (CTR) [78]).

It is also important to emphasize that while there are some real-world systems that target a direct prediction of a relevance score (e.g., rating), in most cases the main goal of RS is to build a good ranked list of items, which is known as the *top-n recommendation task*. This imposes some constraints on the evaluation techniques and model construction. It might be tempting to use and optimize for error-based metrics like root mean squared error (RMSE) or mean absolute error (MAE) due to their simplicity. However, good performance in terms of RMSE does not guarantee equally good performance on generating a ranked list of top-n recommendations [49]. In other words, the predicted relevance score may not align well with the perceived quality of recommendations.

1.2.5 Reproducible results

The problem of reproducibility is closely related to recommendations quality evaluation. Careful design of evaluation procedures is critical for fair comparison of various methods. However, independent studies show that in controlled environments it is problematic to get consistent evaluation results even for the same algorithms on fixed datasets but within different platforms [142].

Situation gets even worse, taking into account that many models tackle similar problems, while using different datasets (sometimes not even publicly available), dif-

ferent data pre-processing techniques [45] or different evaluation metrics. In order to avoid unintended biases, we will focus mostly on the description of the key features of existing methods rather than on a side-by-side comparison of quantitative results.

1.2.6 Real-time recommendations

High quality RS are expected not only to produce relevant recommendations but also respond instantly to system updates, such as new (or unrecognized) users, new items or new feedback [92]. Satisfying this requirement highly depends on implementation: predictive algorithms must have low computational complexity for producing new recommendations and take into account a dynamic nature of real environments. Recomputation of the full RS model in order to include the new entities may take prohibitively long time and users may leave the system without actually seeing any recommendations. This means that RS application should be capable of making incremental updates and also be able to provide instant recommendations at a low computational cost outside of the full model computation cycle. A number of techniques has been developed to fulfill these requirements for the MF case [56, 193, 23]. As it will be shown in Sec. 3.2.2, these ideas can be also applied in the TF case.

1.2.7 Incorporating context information

As has been already mentioned, in real world scenarios interactions between users and items exhibit a multifaceted nature. User preferences are typically not fixed and may change with respect to a specific situation. For example, buyers may prefer different goods depending on the season of the year or time of the day. A user may prefer to watch different movies when alone or when with a group of friends.

We will informally call these situational aspects that shape user behavior a contextual information or *context* for short (see Fig. 1.1). Some other examples of context are location, day of week, mood, the type of a user's electronic device, etc. Essentially, it can be almost anything [13, 46].

Context-aware recommender systems (CARS) can be built with 3 distinct techniques [3]: contextual prefiltering, where a separate model is learned for every context type; contextual postfiltering, where adjustments are performed after a general context-unaware model was built; and contextual modelling, where context becomes

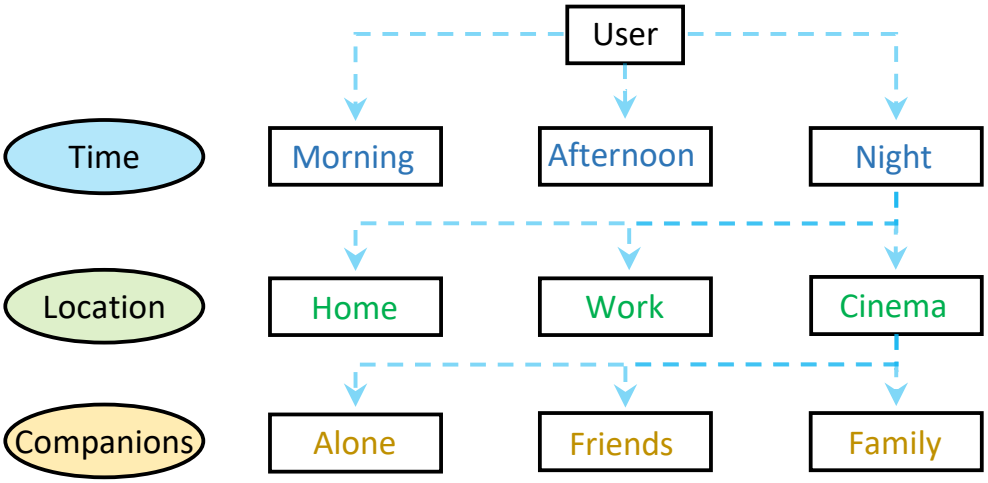

Figure 1.1: Examples of contextual information.

an essential part of the training process. The first two techniques may lose information about interrelations within a context itself. Contextual modelling, in turn, extends the dimensionality of the problem and promotes multi-relational aspect into it. Therefore it may help to achieve more accurate results [85]. Following Eq. (1.1), we can formalize it as follows:

$$f_u : \text{User} \times \text{Item} \times \text{Context}_1 \times \ldots \times \text{Context}_N \rightarrow \text{Relevance Score}, \tag{1.2}$$

where Context_i denotes one of N contextual domains and the overall dimensionality of the model is $N+2$.

As we will see further, TF models fit perfectly into the concept of CARS. With a very broad definition of context, tensor-based methods turn into a flexible tool that allows to naturally model very interesting and non-trivial setups, where the concept of context goes beyond a typical connotation.

1.2.8 Content vs. context

As a precaution, it should be noted that a nonspecifity of a context may lead to interpretability problems. Using a general definition of a context, a content information such as user profile attributes (e.g. age, gender) or items properties (e.g. movie genre or product category) can also be regarded as some type of context (for example, see [85], where age and gender are used to build new context dimensions). However, in practice, especially for TF models, this mixing is typically avoided [184, 138]. One possible reason is a deterministic nature of content information in contrast to what is

usually denoted as a context. Similarly to MF techniques, TF reveals new unseen associations (see Sec. 3.1.3), which in the case of deterministic attributes can be difficult to interpret. It is easy to see in the following example.

For a triplet (*user, movie, gender*) the movie rating may be associated with only one of two possible pairs of (*user, gender*), depending on the actual user's gender. However, once a reconstruction (with help of some TF technique) is made, a non-zero value of rating may now pop-up for both values of gender. The interpretation of such association may become tricky and highly depends on initial problem formulation.

1.3 Quick summary and outlook

Many advantages provided by the collaborative filtering approach not only make it a ubiquitous tool in real-world applications but also draw attention from academia. A considerable part of this focus is specifically devoted to various matrix- and tensor-based factorization methods due to their efficiency, flexibility and relative simplicity. Factorization methods constitute the central part of this thesis work as well. We revisit the general problem formulation and propose several new modifications, which allow to at least partially address the challenges mentioned above.

For example, in Chap. 5 we demonstrate how to improve the warm start scenario with the help of a higher order tensor-based model. The model allows to better represent known user preferences, remove an undesired positivity bias and improve the general user experience. The proposed formulation treats user feedback as an independent categorical variable encoded within its own new dimension. This in many ways resembles the context-aware approach: interactions between users and movies are considered in the context of a certain feedback value.

Furthermore, as it was noted in Sec. 1.2.2, an extreme prevalence of missing data may drastically limit the quality of CF models. Introducing more dimensions can make the problem even worse. The method proposed in Chap. 6 allows to better handle such cases with the help of a new hybrid approach based on a generalization of SVD. It uses side information as an additional source of knowledge for recovering unseen relations. Due to the use of side information, the method is also potentially applicable in the cold start scenario.

In Chap. 7 we show how the previous two approaches can be combined into a unified factorization model. The model allows to preserve the key advantages of both approaches and at the same time circumvents their main shortcomings. It also follows the concept of the content versus context dichotomy, mentioned in Sec. 1.2.8. The context part, drawn from user feedback in our experiments, is encoded within a separate dimension and gets its own latent representation along with users and items. In contrast, the content part is directly encapsulated into the latent feature space of the corresponding entity (user, item or context), which not only respects the aforementioned deterministic nature of real features but also allows to avoid the explosion of the latent space, when the number of real features gets high.

Another important feature shared by all three models is that they offer a straightforward folding-in calculation based on an analytical formula. It, therefore, allows to operate in highly dynamic environments and to almost instantly respond to changes in the system without the need to perform expensive retraining of the models.

Finally, Chap. 8 introduces a new recommendation framework developed during the course of this work. Through an abstraction level, it provides a set of convenient tools for quick prototyping and evaluation of new models. It takes care of many technical aspects, which makes exploration of new ideas a lot easier and also helps to ensure reproducibility of experiments.

Before proceeding to the main part of the thesis, which provides more details on each of the proposed methods, let us first give a broad overview of already existing approaches and techniques, based on either matrix (Chap. 2) or tensor factorization (Chap. 3).

Chapter 2

Matrix Factorization

Matrix factorization is one of the most successful and widely used collaborative filtering techniques. One of the key advantages of MF models is the ability to reduce an initial problem's complexity and provide a compact representation of interaction data generated from observed collective human behavior. With this approach users and items are embedded as vectors in a lower dimensional space of *latent features*. This procedure is known as a *dimensionality reduction* task. As a result, both user tastes and relevant item characteristics can be described by a relatively small set of parameters.

With this representation, the relations between users and items follow general linear algebra rules and vector arithmetic. The utility of a particular item to a particular user can be simply estimated via a scalar product of their vectors in the obtained lower dimensional latent feature space. *This is the key concept that connects various MF models presented in this chapter.* From the geometric point of view, the angle between user and item vectors is smaller for relevant items and is larger for irrelevant ones (see [94, Figure 2] for an illustration). This can be conveniently expressed in terms of the cosine similarity measure and also used for building more efficient neighborhood-based models [2].

Unlike the neighborhood-based techniques, MF is less susceptible to the so-called limited coverage problem [44]. For example, the lack of common preferences information for a pair of users may lead to unreliable correlations in prediction mechanism for the neighborhood models. In contrast, MF models build a more meaningful conceptual description of user interests in terms of latent features, which to a certain extent allows alleviating that problem. A better expressiveness of the MF models also

makes them less sensitive to the data sparsity problem, typically observed in many real applications.

As has been already stated, the concept of *utility*, is one of the key ingredients of MF models. Its purpose is to adequately represent an undetermined decision-making process driven by hidden motives behind a particular user choice. The decision making is indirectly observed via partially available interaction data, expressed in the form of a feedback provided by users to some (not all) items. The goal of any MF approach is, given that data, to estimate the corresponding *utility function* f_u, as defined in 1.1, which will not only agree with the observed part of user preferences but will also help to make predictions on the unobserved part.

In the simplest case the $Relevance\,Score$ can be directly related to the user feedback. Consider a movie recommendation system, where users express how satisfied they are with a certain movie by providing an *explicit* rating value on some Likert scale. The problem of finding f_u can then be transformed into a well studied *matrix completion* problem, which has become especially popular in the recommender systems community after the famous Netflix Prize competition[1]. Even though the rating values are subjective in their nature [8], it is often neglected as the task of recovering the unknown entries of the rating matrix enables many very practical and quite efficient methods of solving the problem of recommendations. A considerable part of Sec. 2.3 is devoted to such methods not only because of their popularity but also as it helps to provide the necessary background for an understanding of more elaborate models.

Standard matrix completion, however, may not be the best choice in the *implicit* case, where the feedback is not intentionally provided by users and is collected via an indirect observation of their actions, such as clicks on product web-pages, amount of product purchases, time spent reading a product description, etc. Note, that the lack of feedback from a user for a particular item does not immediately imply a negative preference, which holds for both explicit and implicit cases. However, in the implicit case, the fact that a user has interacted with an item may not necessarily correspond to a positive preference. Taking that into account requires a more thoughtful problem formulation, which may lead to an alternative definition of the $Relevance\,Score$, abstracted away from the observable feedback (see Sec. 2.3.3). Note that one of the

[1] https://www.netflixprize.com

corner cases of implicit feedback when it simply denotes the fact of interaction is often referred to as One-Class Collaborative Filtering (OCCF) [117, 178].

In addition to that, in many practical applications, it is often more important to return an *ordered list of correctly ranked recommendations*, rather than simply predict rating values. This is known as a *top-n recommendation problem*, where n is the number of recommended items. At first glance, this may seem like a trivial task: once the rating predictions are available, one can simply select the items with the highest predicted score. However, being able to recover rating values accurately does not necessarily guarantee the best performance in terms of generating a ranked list of the most relevant recommendations [92]. This opens the doors for the so-called *learning to rank* models with a substantially different objective (see Sec. 2.4), more coherent with the task of top-n recommendations. Such models are typically not even suitable for the completion task.

As the matrix completion models can be tuned and evaluated in terms of the ranking problem as well, we will distinguish between the two major types of recommendation tasks – the *rating prediction* and the *top-n recommendation* – and provide a view on factorization models through the lens of this distinction where necessary.

2.1 Problem formulation

As has been already noted, the dimensionality reduction approach in recommender systems allows to describe any user preferences and any item characteristics in terms of a small set of model parameters. Along with a compact representation, it also helps to uncover non-trivial patterns within the data and use them to generate meaningful recommendations. Generally speaking, this can be achieved with the help of various methods, such as neural networks, markov decision processes, latent dirichlet allocation and some others. However, in this chapter we focus specifically on the matrix factorization approach.

Let us start from the matrix completion case as it provides a good illustration of some major concepts and serves as a ground for further improvements and generalizations to higher order cases (Chap. 3). Consider an imaginary scenario in which all known users of some recommendation system have provided their preferences for all available items. This can be conveniently represented in the form of a *complete* matrix

of interactions $A \in \mathbb{R}^{M \times N}$. The rows of the matrix correspond to users and its columns correspond to items. Its elements would correspond to some form of a feedback provided by users and this would represent a snapshot of a real "noisy" data. The "noise" may have different nature. It can be caused by variations in individuals' behavior and their tastes or by occasional changes in a context of an interaction, or it can be the result of some other uncontrollable and mostly unpredictable factors. All of it leads to a certain level of unavoidable randomness making the problem of recommendations very complex.

Nevertheless, at a large scale the collective behavior may reveal some regularities and exhibit some common patterns that could be potentially described with a relatively small set of parameters. Therefore, while the dimensionality reduction may lead to a loss of some information, it can still help to uncover and generalize at least some of the hidden commonalities in users' behavior. With this assumption the observed data can be modelled as:

$$A = R + E,$$

where the matrix E denotes the "noise" and R is an approximate *utility matrix* which accommodates the behavioral patterns and have a certain inner structure. *The task of building a recommendation model then translates into the task of recovering* R.

The solution to this problem in the case of a matrix factorization approach can be generally represented in the form of a matrix product:

$$R = PQ^T, \tag{2.1}$$

where matrices $P \in \mathbb{R}^{M \times r}$ and $Q \in \mathbb{R}^{N \times r}$ represent users and items respectively. Each row $\boldsymbol{p}_i^T$ of the matrix P reflects a preference vector of user i, described in terms of r latent features. In other words it gives a representation of user i in the latent feature space. Similarly, each row $\boldsymbol{q}_j^T$ of the matrix Q describes an association of item j with those latent features, i.e. it gives a representation of an item in the latent feature space.

Vectors $\boldsymbol{p}_i$ and $\boldsymbol{q}_j$ are also called an *embedding of users and items onto the latent feature space*. The utility function f_u of an item j to a user i is, therefore, represented by a scalar product $\boldsymbol{p}_i^T \boldsymbol{q}_j$. The number of latent features r is called the *rank of an approximation*. This number is typically much smaller than the number of items or users. Such a representation of a matrix as a product of two other matrices of smaller

sizes is also called a *low rank approximation* and the resulting matrix R is said to have a *low rank structure*.

The final form of the matrices P and Q depends on the formulation of a corresponding optimization problem described in terms of a specific loss function $\mathscr{L}$:

$$\min_{\Theta} \mathscr{L}\left(A, R(\Theta)\right), \tag{2.2}$$

where $\Theta := \{P, Q\}$ is a set of model parameters and $\mathscr{L}$ penalizes deviation of the model from observations. Worth noting here, that the term *deviation should be treated in a broad sense.* As we discussed in the introduction, a particular form of the function $\mathscr{L}$ may go far beyond standard matrix completion formulation (see Sec. 2.4).

Also recall, that in the majority of real systems the observed interactions are typically very scarce and the vast amount of data is missing, which makes the matrix A overly *incomplete*. Therefore, *the optimization problem described by Eq. (2.2) remains ambiguous unless we explicitly define how to deal with the missing values of* A or at least define in what sense a *complete* matrix R approximates an incomplete matrix A. Due to this reason we prefer to avoid the commonly used and intuitive notation $A \approx R$.

We also note, that in some cases an additional processing of the data may help to create a better representation of the observed user behavior and potentially help to improve the quality of recommendations. As an example, in a music recommendation service the logarithmic scaling of a listening frequency (i.e. the number of times a user has listened to a track) may help to generate more accurate recommendations comparing to a naive use of raw counts data as a measure of utility. There is a number of transformation techniques such as data centering and normalization, value binarization, cutting by a threshold, tf-idf transformation and many others which may help to build more accurate recommendation models.

Even in the systems with a fixed explicit feedback, such as a 5-star rating scale, used in many movie recommendation services, a transformation of that scale may improve recommendations. It has an intrinsic connection to a subjective nature of a perceived utility of goods. For example, some users may assign a rating value of 3 to a movie they believe is “OK”, i.e. nothing special but still "watchable", whilst for other users this can be a way to indicate that the movie is completely uninteresting, a total waste of time. In addition to that, some empirical studies show that even for a single user the perceived "distance" between different ratings may vary and the uni-

form rating scale from 1 to 5 used as a measure of a user enjoyment may not be that accurate [8]. All of this, along with the fact that the unobserved data is *missing not at random* (MNAR) [161, 151], may potentially introduce unintended biases in both recommendation models and evaluation measures.

Both described aspects – the way missing data is handled and the choice of a data preprocessing technique – create additional degrees of freedom for a model construction. Sometimes it may directly affect an optimization procedure and lead to very different factorization algorithms. In other cases it may lead to several variations of the same method. In order to explicitly signify the role of these degrees of freedom we will formulate the optimization problem Eq. (2.2) not in terms of an approximation of the matrix A, but rather as an approximation of some function of A:

$$\min_{\Theta} \mathscr{L}\left(T(A), R(\Theta)\right), \tag{2.3}$$

where $T(\cdot)$ denotes a problem-dependent transformation of the data which may include missing values imputation and/or various data preprocessing steps. In the next sections we will cover some of the most famous factorization models resulting from a combination of different data transformation techniques, various loss functions and optimization algorithms.

2.2 SVD-based models

One of the first factorization algorithms used in the field of recommender systems is the singular value decomposition (SVD) [66]. It is a well-established computational tool with efficient implementations in many programming languages, which is included in many modern machine learning libraries and frameworks. SVD is used within a wide range of applications in various domains of data analysis, information retrieval and natural language processing.

Speaking about the latter, SVD has a straightforward relation to the latent semantic indexing/analysis (LSI/LSA) [56, 42]. However, the task of revealing words' semantics based on their occurrence in text documents is in some sense similar to the task of finding alike items based on users' consumption patterns. Not surprisingly, the first SVD-based technique for recommender systems was proposed already in the late 90's, just a few years after this vibrant research field emerged [49].

Generally, early application of SVD in recommender systems had an *enabling* role in a sense that it was used as an intermediate dimensionality reduction step and in order to generate a final list of recommendations its output was fed into a different algorithm based on, for example, a neural network [19] or a nearest neighbors approach [146]. The authors of the latter work also used SVD in a *standalone* regime (with a certain preliminary data normalization) for the *rating prediction* task with a little to no improvement over the competing CF algorithm. However, an even simpler SVD-based model, named *PureSVD* [38], has been later demonstrated to outperform some state-of-the-art algorithms in terms of the *top-n recommendation* task.

We find it necessary to also introduce here some formal definitions and common results from linear algebra, which will help in further explanations. Any *complete* matrix $A \in \mathbb{R}^{M \times N}$ can be represented in the form:

$$A = U \Sigma V^T,$$

where $U \in \mathbb{R}^{M \times M}$ and $V \in \mathbb{R}^{N \times N}$ are orthogonal matrices, their columns are called the left and the right singular vectors respectively; $\Sigma \in \mathbb{R}^{M \times N}$ is a diagonal matrix with non-negative elements $\sigma_1 \geq \ldots \geq \sigma_K$ on its main diagonal called singular values; $K = \min(M, N)$ is a *rank* of SVD. According to the Eckart-Young theorem [48], the truncated SVD of rank $r < K$ with $\sigma_{r+1}, \ldots, \sigma_K$ set to 0 *gives the best rank-r approximation of the matrix A*. NOTE In fact, even though the problem is non-convex, it can be shown that all possible minima in that case are global [160].

2.2.1 PureSVD

Unfortunately, the result of the Eckart-Young theorem cannot be directly applied in recommender systems settings as *SVD is undefined for incomplete matrices*. As a workaround the PureSVD model uses a simple imputation technique: to *replace the missing entries of A with zeroes*. Hence, an incomplete matrix A is transformed into a *sparse* matrix A_0 with zero values inplace of the unknown elements, i.e. $T(A) = A_0$. The corresponding loss function can then be expressed as

$$\mathscr{L}(T(A), R) = \|A_0 - R\|_{\mathrm{F}}^2, \tag{2.4}$$

where $\|\cdot\|_{\mathrm{F}}$ denotes the Frobenius norm. As the loss function is now well defined, we can apply the Eckart-Young theorem to find a *globally optimal* solution to the resulting

optimization task defined by Eq. (2.3):

$$R = U_r \Sigma_r V_r^T, \tag{2.5}$$

where factor matrices $U_r \in \mathbb{R}^{M \times r}$ and $V_r \in \mathbb{R}^{N \times r}$ have orthonormal columns and represent users and items in the reduced latent space with $r \ll \min(M, N)$ distinct latent features. Square diagonal matrix $\Sigma_r \in \mathbb{R}^{r \times r}$ has r largest singular values on its main diagonal. $U \in \mathbb{R}^{M \times r}, V \in \mathbb{R}^{N \times r}$ are orthogonal factor matrices that embed users and movies respectively onto a lower dimensional space of latent (or hidden) features, Σ is a matrix of singular values $\sigma_1 \geq \ldots \geq \sigma_r > 0$ that define the strength or the contribution of every latent feature into resulting score. Equation (2.5) can be equivalently rewritten in the form of Eq. (2.1) simply by allowing $P = U_r \Sigma_r^{\beta}$ and $Q = V_r \Sigma_r^{1-\beta}$, where β is some real number in the interval $[0, 1]$, typically assigned to $1/2$ or 1.

As was noticed by the authors of the PureSVD model, the orthonormality of columns of the factor matrices allows to rewrite Eq. (2.5) in a more convenient form. Assuming that $A_0 = U \Sigma V^T$ is the full SVD of the completed matrix, we have $A_0 V_r V_r^T = U \Sigma V^T V_r V_r^T = U_r \Sigma_r V_r^T$. The last equality is due to the fact that $V^T V_r = [I_r \, 0]^T$, where I_r is the identity matrix of size r and 0 denotes a matrix of all zeros with a conforming size. From here it reads:

$$R = A_0 V_r V_r^T. \tag{2.6}$$

This induces a natural geometrical interpretation: once the right singular vectors are determined, *every row of the prediction matrix R can be computed as an orthogonal projection of the corresponding user preferences onto the latent feature space.* Note, that this eliminates the need for the matrix of user factors U_r, as it can be restored by the means of $U_r \Sigma_r = A_0 V_r$. This can be used to reduce both computational overhead and storage requirements of the model. From now on for brevity we will omit the subscript r in the equations for both matrix- and tensor-based factorizations (Chap. 3), always assuming a low-rank approximation. In other words, we will denote a factor matrix U_r simply as U and likewise for other factor matrices.

Taking into account that in many practical cases a typical sparsity of the matrix is higher than 99%, setting zeros inplace of the missing data introduces a strong bias of the model prediction towards zero values. This makes the model very bad from the matrix completion perspective and totally *unsuitable for the rating prediction*

task. Nevertheless, despite such a bias, the PureSVD approach is known to serve as a *strong baseline in the top-n recommendation problem* outperforming even more elaborate state-of-the-art methods [38, 96]. Of course, it does not imply that PureSVD is always an optimal choice. However, it should be considered by beginner practitioners as a good starting model.

The *truncated SVD* can be computed with the help of an iterative *Lanczos procedure* [66] which invokes a Krylov subspace method and internally uses efficient bidiagonalization techniques supported by a Gram-Schmidt orthogonalization process. The key benefit of such approach is that in order to find r leading singular vectors and corresponding singular values it is only required to provide a rule of how to *multiply an interaction matrix by an arbitrary vector* from the right and from the left. More specifically, given the number of non-zero elements nnz_A of the matrix A that corresponds to the number of the observed interactions, the overall computational complexity of the SVD algorithm can be estimated as $O(nnz_A \cdot r) + O((M+N) \cdot r^2)$ [71], where the first term corresponds to the complexity of a sparse matrix-vector product and the second term is related to an internal orthogonalization process.

2.2.2 Biases and custom data transformation

It has been already noted that user feedback is intrinsically subjective. One of the ways to partially address that subjectivity at least in the rating-based systems is to introduce the concept of the so called user and item bias. *User bias* captures a tendency of a user to systematically assign higher (or lower) ratings depending on how critical the user is in comparison with an average person. Likewise, *item bias* can be described as a tendency of items to receive higher (or lower) ratings. In practice, it turns out that the most part of an interaction "signal" (e.g. rating value) is accommodated by these biases. This allows for even non-personalized recommendation models, called *baseline predictors*, to demonstrate a fairly good prediction quality in the rating prediction task [94].

These biases can be estimated with the help of simple statistics such as an average of user and item ratings calculated over the observed data sample. It is also possible to use more sophisticated estimation methods, e.g. averaging with value damping or even gradient-based optimization [49]. An overall bias b_{ij} (i.e. baseline predictor)

can be expressed as a combination of all systematic biases[2]:

$$b_{ij} = \mu + t_i + f_j, \tag{2.7}$$

where μ is a global bias constant (e.g. global average rating); t_i denotes a user bias or a *tendency* to give higher or lower ratings; in turn f_j reflects an item bias, its *favouredness* or in some sense quality (based on the opinions of raters).

In the PureSVD model these biases can be used as a replacement for the missing data, which therefore reduces the distortion introduced by a straightforward zero-based imputation step and allows to partially address the subjectivity of user preferences. In the simplest scenario one could use Eq. (2.7) to replace the missing entries of A with the corresponding values of the baseline predictor. In this case the transformation T of the data is trivial. It is sufficient to simply subtract the bias values from the known entries of A. The unknown values can then be set to 0, preserving the same sparsity pattern as in A_0. After the preprocessing is done the standard PureSVD model is built on top of the centered data. When generating recommendations the *bias term should be added back* to the predicted scores of the model.

More elaborate data preprocessing techniques can also be supported without sacrificing the computational efficiency. As an example, consider the case when the missing elements of the rating matrix are first filled-in with the values of item average ratings $\boldsymbol{f} \in \mathbb{R}^N$ and then the resulting *complete* matrix is additionally normalized by subtracting user average ratings $\boldsymbol{t} \in \mathbb{R}^M$ [146]. The elements of the obtained centered matrix $T(A) = \hat{A}$ can be expressed as:

$$\begin{cases} \hat{a}_{ij} = a_{ij} - t_i & \text{if } a_{ij} \text{ is known,} \\ \hat{a}_{ij} = f_j - t_i & \text{otherwise.} \end{cases}$$

By construction, the complete matrix $\hat{A}$ is likely to be *dense*. However, it can be split into the sum of a *sparse* matrix $\bar{A}$ with two rank-1 terms (outer products of vectors):

$$\hat{A} = \bar{A} - \boldsymbol{t}\boldsymbol{e}_N^T + \boldsymbol{e}_M \boldsymbol{f}^T, \tag{2.8}$$

[2] The notation we use here slightly differs from what can be commonly seen in the literature – we assign different letters to user and item bias variables, as it helps to avoid an ambiguity in mathematical formulations which involve matrix-vector operations.

where $\boldsymbol{e}_M, \boldsymbol{e}_N$ denote vectors of all ones of a conforming size and the elements of $\bar{A}$ are defined as follows:

$$\begin{cases} \bar{a}_{ij} = a_{ij} - f_j & \text{if } a_{ij} \text{ is known,} \\ \bar{a}_{ij} = 0 & \text{otherwise.} \end{cases}$$

Recall that in order to compute the truncated SVD it is only required to provide a matrix-vector multiplication rule. Multiplying Eq. (2.8) by an arbitrary vector v gives:

$$\hat{A}\,\boldsymbol{v} = \bar{A}\boldsymbol{v} - \boldsymbol{t}\langle \boldsymbol{e}_N, \boldsymbol{v}\rangle + \boldsymbol{e}_M\langle \boldsymbol{f}, \boldsymbol{v}\rangle, \tag{2.9}$$

where $\langle \cdot, \cdot \rangle$ stands for the scalar product of two vectors. Note, that the first term in Eq. (2.9) has the same computational complexity as in the original PureSVD approach as the matrix $\bar{A}$ by construction follows the sparsity pattern as A_0. The last 2 terms are linear with respect to the number of users and items, and therefore the *added complexity is only* $O(M+N)$, which is negligible as it is dominated by the complexity of the standard Lanczos procedure. Moreover, there is clearly *no need to explicitly form the dense matrix* $\hat{A}$ to compute SVD, which allows to avoid unnecessary memory overhead. This technique can be further used for an iterative variant of SVD [88] for achieving a better performance in terms of the rating prediction task.

2.2.3 Handling online updates

Many recommendation services aim to provide an instant engagement for both known users and newcomers as well as quickly update the information about new items in the assortment. In the modern online world with its highly dynamic environment and an overwhelming amount of information this requires the ability to generate recommendations instantly. This, however, would be impossible for large scale recommender systems if the only way to accomplish that would be to recompute the whole model for every new (or unrecognized) user or a newly introduced item.

One common technique designed to support an instant service is called *folding-in* [49], which was initially proposed in the field of information retrieval for the semantic document-term analysis [56]. As long as at least one interaction with a new entity (i.e. user or item) is observed, it allows to approximately update the corresponding latent representation and quickly generate recommendations for this new

entity without the need for the whole model recomputation. Note, that this setting is different from the so called *cold start* regime (Sec. 1.2.1), where no interactions are available.

One of the greatest advantages of the SVD-based approach is an *analytical form of the folding-in*. Unlike many other MF methods it does not require any additional optimization steps to calculate recommendations for a new user/item not present in the training data. Once the latent factors are computed one can use the folding-in formula to generate recommendations without recomputing the whole model. This makes SVD-based models very plausible for use in highly dynamic online environments. For illustration purposes we will consider the new user scenario. New item scenario is trivially obtained by analogy.

Assuming that the model is expressive enough, a new user can be represented with high accuracy as a combination of previously seen users. Therefore, the preference vector $\boldsymbol{a}$ of a new user (with imputed zeroes inplace of the unknowns) can be approximated as $\boldsymbol{a}^T \approx \boldsymbol{u}^T \Sigma V^T$, where $\boldsymbol{u}$ is unknown. Multiplying from the right both parts of this approximate equality by $V\Sigma^{-1}$ and using the orthonormality property $V^T V = I$ one arrives at the following expression:

$$\boldsymbol{u}^T \approx \boldsymbol{a}^T V \Sigma^{-1}. \tag{2.10}$$

This represents an approximate embedding of a new user to the latent feature space. The formula can be further used to directly generate recommendations. By the virtue of Eq. (2.5) one could perform a reverse operation and restore the corresponding *new row* for the matrix R, which after transposing the result reads:

$$\boldsymbol{r} \approx V V^T \boldsymbol{a}, \tag{2.11}$$

where $\boldsymbol{r}$ is a vector of predicted relevance scores. Provided that there are k items in the preference vector $\boldsymbol{a}$, the overall complexity of generating recommendations for a single user is $O(Nkr)$, which is the result of the chain of matrix-vector multiplications.

From the geometrical point of view, Eq. (2.11) can be treated as an *orthogonal projection* of user preferences onto the space of latent features represented by V. Comparing this result to Eq. (2.6) suggests that it can be used to *generate recommendations for both known and new users*. All it requires is a list of user preferences. This also means that one can generate recommendations based on any combination of items

even if it does not correspond to any particular known user. In the latter case it gives an estimate of possible user preferences, implicitly relying on the assumption that the learned model is expressive enough. In turn, for the known users it corresponds to the exact prediction formula.

As a precaution remark, the folding-in approach is only approximate and leads to the loss of orthogonality of factors. In the long run it accumulates an error and once in a while it is advised to fully recalculate the model, especially if a lot of new data is collected. Alternatively, *incremental update* techniques can be employed in order to avoid expensive recomputations [17, 194, 23].

2.2.4 The family of eigendecomposition algorithms

The PureSVD model can be viewed as a member of a broader family of eigendecomposition algorithms. Consider an SVD-based approximation $\tilde{A} \approx U\Sigma V^T$ for some complete matrix $\tilde{A}$ with standardized data. The corresponding correlation matrix $\tilde{A}^T\tilde{A} \approx V\Sigma^2 V^T$ would represent the well known PCA with principal components given by $\tilde{A}V = U\Sigma$. The principal components can be then utilized to indicate similarity between users (or items in the transposed case) and build a neighborhood-based recommender system.

This path was initially explored by the authors of the *Eigentaste* model [64] designed for the jokes recommendation system. The authors selected a subset (called the gauge set) of the observed data, where only items rated by all users were present. This has led to a *complete dense matrix* of ratings A. The only transformation $T(A)$ the authors used on top of it was the standardization of rating values, allowing to build a Pearson correlation matrix and apply classical PCA. The authors used the first 2 principal components and a clustering technique in this lower dimensional space in order to group like-minded users. In every group (or cluster) the rating for every *non-gauge* item was estimated as a mean value averaged across those users of the group who has provided rating for this item. As for the new users, they were requested to firstly provide ratings on the gauge items. After that the ratings were projected to the lower dimensional space allowing to assign the newcomers to the known clusters and generate averaging-based recommendations similarly to the known users.

Note, that rating predictions generated by Eigentaste are not fully personalized as they are assigned to a whole cluster of users at once. Moreover, the requirement of the dense gauge set can be fully satisfied only in specific environments with sufficiently large amount of user feedback and/or relatively small number of items to interact with (which is exactly the case with the jokes dataset used in the work). In many real-world settings with very high sparsity of the data these can be difficult or even impossible to guarantee. However, it turns out, that at least in the case of top-n recommendation task such restrictions can be alleviated. As shown by the authors of the *EIGENREC* model [112], as long as the rating prediction is not one of the goals of a recommender system, one could build a more flexible and more general approach following the paradigm of PureSVD.

The authors make the following observation: *PureSVD can be viewed as an eigendecomposition of a scaled user-based or item-based cosine similarity matrix.* For instance, in an item-based case it solves an eigendecomposition problem for the following matrix cross-product:

$$A_0^T A_0 \equiv DCD \approx V \Sigma^2 V^T, \tag{2.12}$$

where the scaling matrix $D \in \mathbb{R}^{M \times M}$ is diagonal with diagonal elements $d_{ii} = \|\boldsymbol{a}_i\|_2$ and $\boldsymbol{a}_i$ denotes the ratings of the item i encoded within the i-th column of the matrix A_0. Each element c_{ij} of the symmetric matrix $C \in \mathbb{R}^{M \times M}$ equals to the cosine similarity between item i and item j:

$$c_{ij} = \cos(i, j) = \frac{\boldsymbol{a}_i^T \boldsymbol{a}_j}{d_{ii} d_{jj}}. \tag{2.13}$$

From here it follows, that by altering the scaling factors D and/or by replacing C with some other inter-item proximity or correlation matrix S:

$$DCD \to D^p S D^p,$$

one can obtain a new model with a different inner structure of the latent space. Here p is some real number (the authors used values in the range [-2, 2]) and S is a new proximity matrix, which can be based on Pearson correlation, Jaccard index or many other similarity measures. The authors emphasize, that in fact even *the choice of a scaling factor may have a significant impact* on the quality of recommendations. This

scaling allows to control the sensitivity of the model to the popularity of items, and therefore to some extent mitigates the problem of unbalanced observation data present in the majority of recommender systems.

Similarly to PureSVD the authors use the Lanczos procedure in order to build an orthogonal basis. They propose their own parallel and highly efficient implementation of it. Therefore, the EIGENREC approach allows to preserve the benefits of PureSVD which include a good scalability and a quick way to generate recommendations according to Eq. (2.11) for both known and newly introduced users. The approach also gives more flexibility comparing to the standard PureSVD model and unlike the Eigentaste model allows to operate on the full assortment of items from the very beginning. It provides an instrument for a more intricate tuning, potentially making it suitable for a wider class of problems.

2.3 Weighted low-rank approximation

A straightforward data imputation is not the only way of dealing with missing values. Alternatively, one could try to avoid making any strict assumptions on the missing values and either ignore them completely or introduce some confidence-based description of it. Indeed, the fact that some interactions between users and items are unobserved does not immediately suggest that these interactions will never happen. For example, a user may never interact with an item simply due to inability to notice it among many other similar items in a large assortment. On the other hand, if a user consumes one item more often than another one, it may increase our confidence that the item is more relevant or more interesting for a user.

Hence, bringing the concept of a confidence-based weighting for both observed and unobserved interactions into a factorization model may help to create more accurate recommender systems. A common way to express the corresponding loss function reads:

$$\mathscr{L}\left(T(A), R\right) = \left\| W \circ \left(T(A) - R\right)\right\|_{\mathsf{F}}^2, \tag{2.14}$$

where $W = \left[\sqrt{w_{ij}}\right]$ is a matrix of non-negative weights $w_{ij} \geq 0$ and $\circ$ denotes *Hadamard product*, i.e. an elementwise multiplication between two matrices. The weight values of W typically depend on the observed data $W = W(A)$. We take the square root of weights w_{ij} in order to conform with an equivalent elementwise for-

mulation of the loss function:

$$\mathscr{L}\left(T(A), R\right) = \sum_{i,j} w_{ij}(a_{ij}^{(T)} - r_{ij})^2, \tag{2.15}$$

where $a_{ij}^{(T)}$ denotes an element of the matrix $T(A)$ at the intersection of the i-th row and j-th column.

One of the most popular choices of the weights is based on $\{0, 1\}$ values simply indicating the fact of interaction. The corresponding binary weight matrix W is then defined by:

$$\begin{cases} w_{ij} = 1 & \text{if } a_{ij} \text{ is known,} \\ w_{ij} = 0 & \text{otherwise.} \end{cases} \tag{2.16}$$

With this formulation, no data imputation is required as *all missing elements of the matrix A are simply ignored.* More elaborate weighting schemes are discussed in Sec. 2.3.3.

Typically, the number of users and items is very large while at the same time the number of observed interactions between them is very small. Therefore, the model obtained as a result of minimization of the loss function defined by Eq. (2.14) is likely to overfit and produce poor prediction quality on the unobserved part of the data. In order to prevent this overfitting additional constraints are typically imposed on the parameters of the model. Most commonly, a simple regularization is used for that purpose leading to the following regularized optimization objective:

$$\mathscr{J}(\Theta) = \mathscr{L}(\Theta) + \Omega(\Theta), \tag{2.17}$$

where $\mathscr{L}(\Theta)$ is defined by Eq. (2.14) (we omit the full notation of the input arguments for brevity) and $\Omega(\Theta)$ is some regularization function typically expressed in terms of some vector or matrix norm. Many factorization models use a simple quadratic term, allowing to penalize an undesired growth of the parameters' values:

$$\Omega(\Theta) = \lambda\left(\|P\|_{\mathrm{F}}^2 + \|Q\|_{\mathrm{F}}^2\right), \tag{2.18}$$

where $\lambda > 0$ is an additional model's hyper-parameter called regularization coefficient. In some cases a separate value is assigned to each factor matrix for more granular tuning of the model, which sometimes helps to achieve a better prediction quality. Altering the regularization function may also help to induce a specific structure on the resulting latent space, e.g. one could use l_1 norm to obtain sparse latent factors. In

some other cases, when the data is strictly non-negative, imposing a non-negativity constraint on the factors helps to avoid meaningless predictions and may also improve generalization. In the case of binary input data it has a meaning of soft clustering or "fuzzy membership" [173].

2.3.1 Optimization techniques

A recommendation model is learned as a solution to the corresponding optimization problem:

$$\Theta^* = \underset{\Theta}{\operatorname{argmin}} \mathscr{J}(\Theta). \tag{2.19}$$

This can no longer be directly solved with the help of classical SVD and alternative optimization methods are required. Some of the most popular options are gradient-based methods, especially the *stochastic gradient descent* (SGD) [22], and alternating minimization methods such as *alternating least squares* (ALS) [201] and *coordinate descent* (CD) [190].

In general, these methods no longer guarantee global convergence and, therefore, the optimization requires careful initialization and hyper-parameters tuning. Nevertheless, the methods in practice exhibit fairly good convergence behavior which makes them the main building blocks for many recommender models. More advanced optimization techniques based on *Riemannian optimization* [177] also seem promising in recommender systems settings, offering quick convergence and high scalability in low-rank approximation tasks [190].

Gradient-based techniques

The main idea of the gradient-based approach (also called batch gradient) is to iteratively make steps in the direction that is opposite to the gradient of the optimization objective. Each iteration step in its naive implementation is based on the following sequential update rule for the model parameters:

$$\begin{cases} \boldsymbol{p}_i \leftarrow \boldsymbol{p}_i - \eta \dfrac{\partial \mathscr{J}}{\partial \boldsymbol{p}_i}, \\ \boldsymbol{q}_j \leftarrow \boldsymbol{q}_j - \eta \dfrac{\partial \mathscr{J}}{\partial \boldsymbol{q}_j}, \end{cases}$$

where η is a step size also called learning rate; its value can be a constant real number determined empirically with cross-validation or, in more advanced cases, depend on iterations.

The algorithm makes a full pass through all observations, called *epoch*, in order to perform a full update of matrices P and Q. Iterations continue until the maximum number of epochs is reached or a convergence criteria is met. Note that finding the gradient at each iteration can be quite computationally demanding and suffers from many redundant calculations. At large scale this may lead to both slow convergence and high memory load.

A more efficient implementation, which is the essence of SGD, is to approximate a full gradient with the gradient computed over a single observation or a small group of them (called mini-batch). Such smaller updates are easier to find at the cost of a less straightforward convergence. This allows to sweep through the entire dataset in a single pass for the full update of parameters and has a very low memory footprint. In the case of a single observation update, the update rules are as follows:

$$\begin{cases} \boldsymbol{p}_i \leftarrow \boldsymbol{p}_i + \eta(e_{ij}\boldsymbol{q}_j - \lambda \boldsymbol{p}_i), \\ \boldsymbol{q}_j \leftarrow \boldsymbol{q}_j + \eta(e_{ij}\boldsymbol{p}_i - \lambda \boldsymbol{q}_j), \end{cases} \tag{2.20}$$

where $e_{ij} = a_{ij}^{(T)} - r_{ij}$ measures how off is the prediction of the model at the current step from the ground-truth.

The method strongly depends on initialization of its parameters, performed at the beginning. A quite common practice is to use a normal distribution with zero mean and small deviation. It is also advised to shuffle the data prior to optimization in order to avoid unintended biases in the resulting model. The overall complexity of the approach is $O(nnz_A \cdot r)$.

Note that *SGD is inherently incremental*, which gives an "out-of-the-box" equivalent of the folding-in technique for the model updates. For example, in the case of a newly introduced user with at least a few known preferences one can simply iterate over these preferences with the first line of Eq. (2.20) until it converges. The other parameters related to items stay fixed in that case. New items can be handled in a similar fashion.

Alternating minimization techniques

In turn, the ALS-based methods decompose the optimization task into the sequence of the least squares problems. Note that while the optimization problem defined by Eq. (2.19) is non-convex, it is *bi-convex* with respect to its parameters. In other words, for fixed P it is convex in Q and for fixed Q it is convex in P. Moreover, the optimization problem can be solved independently for every row of P and Q. Therefore, one can iteratively minimize the objective function by switching between user and item factors and updating their rows as follows:

$$\begin{cases} \boldsymbol{p}_i \leftarrow \underset{\boldsymbol{p}_i}{\operatorname{argmin}} \mathscr{J}(\Theta), \\ \boldsymbol{q}_j \leftarrow \underset{\boldsymbol{q}_j}{\operatorname{argmin}} \mathscr{J}(\Theta). \end{cases} \tag{2.21}$$

After each iteration the objective function is guaranteed not to increase. However, unlike the unweighted case, there are no global guarantees for convergence in general. In practice, the algorithm is reported to require only around 10 or slightly more epochs to achieve a good approximation [14, 82].

In order to find the update rules for Eq. (2.21), it is convenient to rewrite both $\mathscr{L}$ defined by Eq. (2.14) and Ω defined by Eq. (2.18) in the row-wise and column-wise forms, corresponding to $\boldsymbol{p}_i$ and $\boldsymbol{q}_j$ respectively. For example, in the user-wise case it reads:

$$\mathscr{J}(\Theta) = \sum_i (\boldsymbol{a}_i - Q\boldsymbol{p}_i)^T W^{(i)} (\boldsymbol{a}_i - Q\boldsymbol{p}_i) + \lambda \sum_i \boldsymbol{p}_i^T \boldsymbol{p}_i + \lambda \|Q\|_{\mathrm{F}}^2, \tag{2.22}$$

where $W^{(i)} = \operatorname{diag}\{w_{i1}, w_{i2}, \ldots, w_{iN}\}$ is a diagonal matrix of weights and $\boldsymbol{a}_i$ is the *i-th row* of the matrix $T(A)$, i.e. it represents the *preference vector of user i with respect to all items*. After finding the derivative $\partial \mathscr{J} / \partial \boldsymbol{p}_i$ and setting it to zero one arrives at the following equation for $\boldsymbol{p}_i$:

$$\left(Q^T W^{(i)} Q + \lambda I\right) \boldsymbol{p}_i = Q^T W^{(i)} \boldsymbol{a}_i. \tag{2.23}$$

This gives a standard linear system of equations with the $r \times r$ symmetric positive definite matrix $\left(Q^T W^{(i)} Q + \lambda I\right)$. Direct solution of the system can be found in $O(r^3)$ time, for example, by the means of Cholesky decomposition. The resulting expression for the $\boldsymbol{p}_i$ update reads:

$$\boldsymbol{p}_i \leftarrow \left(Q^T W^{(i)} Q + \lambda I\right)^{-1} Q^T W^{(i)} \boldsymbol{a}_i. \tag{2.24}$$

Due to the symmetry of the objective function, in order to find an update rule for $\boldsymbol{q}_j$ one can simply replace $Q, W^{(i)}$ and $\boldsymbol{a}_i$ with their corresponding counterparts:

$$\boldsymbol{q}_j \leftarrow \left(P^T \bar{W}^{(j)} P + \lambda I\right)^{-1} P^T \bar{W}^{(j)} \bar{\boldsymbol{a}}_j, \tag{2.25}$$

where $\bar{W}^{(j)} = \text{diag}\{w_{1j}, w_{2j}, \ldots, w_{Mj}\}$ and $\bar{\boldsymbol{a}}_j$ denotes the *j-th column* of the matrix $T(A)$, i.e. the *preference vector of all users against item* j. At each epoch the algorithm updates all rows of the matrices P and Q, which can be done in parallel. As in the SGD case, the iteration process repeats until either the number of epochs exceeds some threshold value or the objective function ceases to decrease (with respect to a predefined tolerance). The overall complexity of the algorithm is estimated as $O(nnz_A \cdot r^2 + (M+N)r^3)$ [124].

Note, that the same update rules can also be used to calculate approximate predictions for the *new entities*. Indeed, as every update is just the solution of the corresponding least squares problem, one can replace $\boldsymbol{a}_i$ or $\bar{\boldsymbol{a}}_j$ with the preference vector of a newly introduced user or item respectively. This technique is similar to the *folding-in* update used in PureSVD.

Another important consideration is that the time, required to solve Eq. (2.23), can be further reduced with additional computational tricks. For example, the straightforward application of the *Sherman-Woodbury-Morrison* formula gives an analytic expression for incremental calculations of the matrix inverse at each iteration. This, however, may not always provide a considerable speed-up and highly depends on the data sparsity [124].

Alternatively, instead of the direct approach one could use iterative linear system solvers in order to find an approximate solution. A worth noting candidate is the *conjugate gradient* (CG) method [66], which is closely related to the Lanczos process and similarly requires only matrix-vector multiplications for performing the task. The method allows to reduce the complexity of the matrix inverse computation to $O(r)$ instead of $O(r^3)$ as in the original approach. It gives a decent trade-off between the accuracy of each individual update and the overall convergence speed [171] and works quite well in practice[3].

[3]Its open-source implementation available at https://github.com/benfred/implicit is shown to provide a remarkable speedup almost without the drop in quality.

Coordinate descent

Another iterative approach for performing the optimization task is to employ the (block) coordinate descent method (CD) [18, Section 2.7]. In the context of the low-rank approximation of complete matrices it was explored in [35], where the authors additionally consider nonnegativity constraints. A few efficient variations of the method were also proposed for the missing value estimation in the recommender systems with explicit feedback [15, 124, 190]. Recently, several efficient implementations were also proposed for the OCCF case [189, 12].

Generally, instead of the bulk update of latent feature matrices performed in ALS, CD successively updates either blocks of variables (e.g. rows or columns of the factor matrices) or simply a single variable. Such formulation leads to a convex optimization and avoids computation of a matrix inverse. For example, by declaring the result of an update in the variable p_{ik} as θ one arrives at the following optimization subproblem:

$$f(\theta) = \sum_{ij} w_{ij} \left(a_{ij} - \left(\boldsymbol{p}_i^T \boldsymbol{q}_j - p_{ik} q_{jk} \right) - \theta q_{jk} \right)^2 + \lambda \theta^2, \tag{2.26}$$

where $f(\theta)$ is a univariate quadratic function. Its optimum value is then given by:

$$\theta^* = \frac{\sum_j w_{ij} \left(a_{ij} - \boldsymbol{p}_i^T \boldsymbol{q}_j + p_{ik} q_{jk} \right) q_{jk}}{\lambda + \sum_j w_{ij} q_{jk}^2}. \tag{2.27}$$

Similar expression can be obtained for updates in the matrix Q. This approach also offers a trade-off. The algorithm may require more epochs to converge, however, each iteration within every epoch becomes much cheaper. Despite being less popular than ALS and SGD, CD offers a competitive quality of recommendations with a number of computational advantages [190].

2.3.2 Biased matrix factorization

As was already noted in Sec. 2.2.2, the rating prediction quality can be improved with the concept of biases, which absorb a significant part of the feedback signal. A similar data transformation procedure with manually crafted biases can be applied for the weighted MF problem as well. However, unlike the SVD-based case, the weighted

formulation of the problem is more flexible and allows to declare bias variables as additional model parameters [121, 94].

With this approach, the predicted value of the rating r_{ij} assigned by user i to item j is modelled as follows:

$$r_{ij} = \mu + t_i + f_j + \boldsymbol{p}_i^T \boldsymbol{q}_j, \tag{2.28}$$

where all bias variables $\{t_i\}$ and $\{f_j\}$ are *learned along with other model parameters*, i.e. $\Theta = \{\boldsymbol{t}, \boldsymbol{f}, P, Q\}$. The global average μ is usually pre-estimated based on the known values of ratings. One may conveniently rewrite the prediction formula in a compact matrix form, following the outer product rule similarly to Eq. (2.8):

$$R = \mu E + \bar{P}\bar{Q}^T,$$

where the block matrices $\bar{P} = [\boldsymbol{t}\ \boldsymbol{e}_M\ P]$ and $\bar{Q} = [\boldsymbol{e}_N\ \boldsymbol{f}\ Q]$ have a particular form of the first two columns comprised by the bias vectors and vectors of all ones; $E = \boldsymbol{e}_M \boldsymbol{e}_N^T$ is an $M \times N$ matrix of all ones.

Clearly, shifting the data values by μ would give similar to Eq. (2.1) form. However, the bias terms increase an overall rank of the solution by 2. Moreover, the result does not correspond to an arbitrary unbiased MF model of rank $r + 2$ due to a certain structure of the first 2 columns in the factor matrices. In some sense biases can be viewed as a specific constraint on the factors, which is used to reflect the core assumption about the underlying rating mechanism.

The SGD-based variation of this matrix factorization approach became popular after it was published in the famous blog post [4] by Simon Funk, when he attended the Netflix Prize competition. Due to that, sometimes this algorithm is also called *FunkSVD*. It has become an internal part of many other MF algorithms. The full update rule, including additional bias updates, reads:

$$\begin{cases} \boldsymbol{p}_i \leftarrow \boldsymbol{p}_i + \eta(e_{ij}\boldsymbol{q}_j - \lambda \boldsymbol{p}_i), \\ \boldsymbol{q}_j \leftarrow \boldsymbol{q}_j + \eta(e_{ij}\boldsymbol{p}_i - \lambda \boldsymbol{q}_j), \\ t_i \leftarrow t_i + \eta(e_{ij} - \lambda t_i), \\ f_j \leftarrow f_j + \eta(e_{ij} - \lambda f_j). \end{cases}$$

[4] http://sifter.org/simon/journal/20061211.html

As a practical remark, such a representation via the bias terms also allows to quickly estimate the rating values for previously unobserved items or users with no associated ratings. In that case it falls back to the baseline value contained in the corresponding bias term and there is no contribution of the factorization part. This estimate can be further improved after at least one rating value is provided into the system with the incremental approach similarly to the unbiased MF case.

2.3.3 Confidence-based models

Another important example of the weighted matrix factorization approach is based on a more flexible treatment of both observed and unobserved interactions. Consider the case, where users exhibit different behavior depending on how much they like a particular item. For example, when a user plays a particular sound track several times while skipping some other track just after listening to the first 10 seconds, this would be a clear indication that the first track is more interesting for the user. In other words, our *confidence* that the user enjoys the music is higher in the case of the first track. Likewise, the fact that user has never played some track does not immediately suggest that the track is not interesting - the user may be simply unaware of it. However, our confidence in the relevance of the track is lower in this case.

In order to account for such an uncertainty it seems reasonable to associate some confidence measure with every possible interaction. Instead of simply ignoring the missing data and assigning constant weights to the known interactions as in Eq. (2.16) we would like to change the weights of interactions depending on various conditions and to treat both observed and unobserved data in a more thorough way. The general form of a confidence-based loss function is slightly different from Eq. (2.14):

$$\mathscr{L}\left(T(A), R\right) = \|W(A) \circ (S - R)\|_{\mathrm{F}}^2. \tag{2.29}$$

where the binary matrix $S \in \mathbb{B}^{M \times N}$ with elements

$$\begin{cases} s_{ij} = 1, & \text{if } a_{ij} \text{ is known,} \\ s_{ij} = 0, & \text{otherwise} \end{cases} \tag{2.30}$$

indicates whether a particular interaction has occurred. The weights matrix $W = [\sqrt{w_{ij}}]$ encodes a confidence in the observed feedback and *directly depends on the values of A*.

Note that this approach is not designed to predict an exact rating value. It rather focuses on the prediction of a probability of a certain event taking into account an additional information, be it a rating value, a browsing behavior or any other form of an explicit or implicit feedback that allows to quantify the corresponding confidence level. A particular choice of the confidence measure may significantly impact the performance of a recommendation model. A few different techniques were proposed independently by several research groups [82, 117]. The substantial difference in the proposed models is in the way the weighting is applied.

The authors of the so called *Weighted Regularized Matrix Factorization* model (WRMF) [82], sometimes also called *implicit ALS* or *iALS*, propose to assign constant weight of 1 to the unobserved interactions, and increase the weight for any observed interaction proportionally to a satisfaction of a user with an item estimated from the expressed feedback:

$$w_{ij} = 1 + \alpha g\left(a_{ij}^{(T)}\right), \tag{2.31}$$

where α is an empirically determined coefficient of proportionality. The estimation function g is the most subjective part of the model and may vary depending on the domain of application and the type of available data. The authors give a few examples of it based on linear $g(x) = x$ and logarithmic $g(x) = \log(1 + \frac{x}{\epsilon})$ approximation (with an extra tuning parameter ϵ), which work well in practice.

The authors propose to use ALS optimization as it allows to efficiently handle computations with complete matrices S and W. The general form of the solution stays the same as in standard ALS:

$$\begin{cases} \boldsymbol{p}_i \leftarrow \left(Q^T W^{(i)} Q + \lambda I\right)^{-1} Q^T W^{(i)} \boldsymbol{s}_i, \\ \boldsymbol{q}_j \leftarrow \left(P^T \bar{W}^{(j)} P + \lambda I\right)^{-1} P^T \bar{W}^{(j)} \bar{\boldsymbol{s}}_j, \end{cases} \tag{2.32}$$

where $W^{(i)} = \text{diag}\{w_{i1}, w_{i2}, \ldots, w_{iN}\}$, $\bar{W}^{(j)} = \text{diag}\{w_{1j}, w_{2j}, \ldots, w_{Mj}\}$; $\boldsymbol{s}_i$ is a *binary* preference vector of user i with respect to all items, and $\bar{\boldsymbol{s}}_j$ is a *binary* preference vector of all users against item j. The authors came up with an elegant computational trick wich allows to avoid redundant computations making the algorithm highly scalable.

Alternatively, the authors of the second approach, referred to as *weighted ALS* or *wALS* [117], propose to assign the constant weight value of 1 for all known observations and in contrast to WRMF use alternate weighting schemes for the unobserved part. The weighting scheme can be based either on small constant values in the range [0, 1] or on some data aggregation which takes into account popularity effects. For example, in the user oriented approach the weights for the *unobserved* data are proportional to the number of ratings provided by user. The rationale behind is that the higher is the number of ratings provided by user, the more likely it is that the remaining non-rated items are irrelevant for that user. Likewise, in the item oriented case the lower popularity of an item would increase the corresponding weights for negative (unobserved) interactions.

2.3.4 Combined latent representations

A high level intuition behind latent features is often provided in terms of the ability to capture intrinsic item properties as well as user motivation and interests. Latent features are often treated as indicators of some user tastes and items' affinity to them. However, in practice, it can be quite difficult to directly map a single real feature to its latent representation [174]. It is more likely that each latent feature will instead characterize some tangled combination of various aspects.

Moreover, the number of these aspects can be large and they may have a complicated, multifaceted nature making it hard to interpret them by a virtue of standard parametrization. On the other hand, this information may play an important role in the decision making process. Ignoring it may not only limit the expressiveness of a recommendation model, but also hinder its ability to uncover valuable implicit relations within the observed data.

One of the ways to improve sensitivity of a model to a multi-aspect input is to explicitly impose an aspect-based structure on the latent representation of users and items. As an example, consider an online retail shop where customers tend to purchase only a few items and rarely provide an explicit feedback. This would lead to a very sparse interaction matrix and make the decision making process obscure for a recommendation model.

Meanwhile, it is typically possible to collect additional information such as what pages users visit during their search for a product, what information they look for, what products they consider together, etc. Including such information into a model allows to increase an understanding of user interests, and therefore help to create a better prediction model. With the flexibility of a weighted matrix factorization this can be achieved directly by adjusting the optimization objective.

One of the earliest examples of such approach is the *NSVD* model [121], where every user is characterized by a combination of items he or he interacted with. It can be especially helpful in the case of extreme sparsity and the lack of any side information, giving a more "smooth" representation of the data. The author of the model proposed 2 variations of such representation: based on binary vectors (simply denoting the fact of interaction) and based on latent features of items. In the latter case the solution can be sought in the following form:

$$R = SQQ^T \tag{2.33}$$

where S is a sparse matrix of aggregation coefficients with binary elements defined simialrly to Eq. (2.30). Here we omit biases as they can be trivially added.

The matrix product SQ in Eq. (2.33) gives an aggregated representation of every user via the latent features of consumed items. As a result, the contribution into the prediction score is defined by all the actions taken by the user, independently of the user-assigned rating values. Note that the model has a reduced number of parameters which can be especially suitable in the systems with very large amount of users and may potentially help to avoid a certain redundancy. It also has inspired further research in this direction and has led to more elaborate models such as *SVD++* and *Asymmetric-SVD* [93]. Later it was shown to be a special case of the more general models, namely *SVDFeature* [30] and *Factorization Machines* [132].

In the SVD++ model, which turned out to provide results superior to Asymmetric-SVD, the latent features of users are not replaced, but rather are augmented with an additional information about multiple aspects of user-item interactions in the following way:

$$R = (P + \bar{S}L)Q^T, \tag{2.34}$$

where L represents an independent of Q latent subspace, which is used to build neighborhoods of items rated together by the same user. The sparse aggregation matrix $\bar{S}$

has the same sparsity pattern as S. In contrast to NSVD, its values are not binary and are row-normalized, so that the norm of every row would be equal to 1, i.e. $\bar{S} = D^{-1}S$, where $D = \text{diag}\{\|\boldsymbol{s}_1\|_2, \|\boldsymbol{s}_2\|_2, \ldots, \|\boldsymbol{s}_M\|_2\}$ and $\boldsymbol{s}_i$ is an i-th row of S. Such normalization prevents susceptibility of the model to popularity of items and to contribution of very active raters.

Note that multiple types of feedback can be easily incorporated into the model simply by adding more aggregation terms, i.e. $P + \bar{S}_1 L_1 + \bar{S}_2 L_2 + \ldots$, corresponding to different types of feedback (e.g. purchase activity, browsing history, etc.). The key drawback of such approach is an increased number of parameters, which makes the model more difficult to train and prone to overfitting.

Eq. (2.34) can be reformulated as $R = (X\bar{P})Q^T$, where $X = [I\, \bar{S}]$ and $\bar{P}^T = [P^T L^T]$ are block matrices of aggregation coefficients and joint latent features respectively. Up until now we have used coefficients matrix X to combine items rated by the same user. However, it can also be used to reflect any sort of additional information which helps better describe the observed interactions. For example, instead of (or along with) indicating the rated-together items, it can be used to encode relevant user attributes and group users with respect to these attributes. The matrix $\bar{P}$ will be extended with the corresponding embeddings of these attributes onto the latent feature space similarly to how it was performed for items with the matrix L.

The same reasoning can be applied with respect to the matrix Q which can be replaced with an aggregated view on different item properties and the item-related interaction aspects. The most general formulation of such representation can be compactly described as:

$$R = XP(YQ)^T, \tag{2.35}$$

where the block matrices $P^T = [P_1^T\, P_2^T \ldots]$ and $Q^T = [Q_1^T\, Q_2^T \ldots]$ now represent various user-based, item-based and mutual aspects of the observed interactions. Sparse coefficient matrices $X = [X_1\, X_2 \ldots]$ and $Y = [Y_1\, Y_2 \ldots]$ with the corresponding block structure allow to aggregate various latent vectors to represent every interaction from a multi-aspect perspective. This aggregated model is known as *SVDFeature* [30]. Due to its ability to take side information into account it can be considered as a representative of the so called *hybrid approach* (see Sec. 1.1.3).

There is one nuance that is worth noting here. The authors of the model propose to represent the global bias as a weighted sum of global biases calculated with respect

to different aspects. It is more convenient to demonstrate it with an equivalent to Eq. (2.35) elementwise formulation, now including all bias terms:

$$r_{ij} = b_0 + \boldsymbol{t}^T \boldsymbol{x}_i + \boldsymbol{f}^T \boldsymbol{y}_j + \boldsymbol{x}_i^T P Q^T \boldsymbol{y}_j, \tag{2.36}$$

where $b_0 = \sum_{g \in G} \gamma_g \mu_g$ is a global bias aggregated over the group of aspects denoted by G with individual weight coefficients γ_g and bias values μ_g.

Note that the bilinear form of Eq. (2.36) can be viewed as a special case of a polynomial expansion:

$$r(\boldsymbol{z}) = b_0 + \boldsymbol{b}^T \boldsymbol{z} + \boldsymbol{z}^T H \boldsymbol{z} + \dots \tag{2.37}$$

The connection to the SVDFeature model can be seen with the following substitution: $\boldsymbol{b}^T = [\boldsymbol{t}^T \, \boldsymbol{f}^T]$ and $\boldsymbol{z}^T = [\boldsymbol{x}^T \, \boldsymbol{y}^T]$, where $\boldsymbol{x}^T$ and $\boldsymbol{y}^T$ are some rows of the matrices X and Y. The coefficients vector $\boldsymbol{z}$ now encodes the full information about an interaction between some user and some item with respect to all related aspects[5], as was discussed previously. Hence, the quadratic term $\boldsymbol{z}^T H \boldsymbol{z}$ with symmetric positive semi-definite matrix H allows to account for an interplay between any entities and any aspects in their contribution to the final prediction score. Note that H subsumes matrices P and Q in a certain way and the parameters of the model are described as $\Theta = \{b_0, \boldsymbol{b}, H\}$.

Such a generalization leads to the next hybrid approach and a popular machine learning algorithm, namely *Factorization Machines* (FM) [132], which has been proven to perform well in recommender systems. The author of the model notes that the matrix H should have a low-rank structure in order to deal with the sparsity problem and increase the expressiveness of the model:

$$H = V V^T$$

where V embeds all users, items and the corresponding side information onto the lower dimensional latent feature space. In addition to that, all self-influence terms (i.e. x_i^2) are excluded and the symmetry of the model (i.e. the equivalent contribution of both $x_i x_j$ and $x_j x_i$ interplay terms) is taken into account, which produces the following relevance score function:

$$r(\boldsymbol{z}) = b_0 + \sum_i b_i z_i + \sum_i \sum_{j=i+1} \langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle z_i z_j \tag{2.38}$$

[5]In the case of categorical data, e.g. user or item id, user gender, movie genre, etc., this method of building a sparse representation of the multidimensional input data is called *one hot encoding*.

The task of generating recommendations, therefore, boils down to solving the polynomial regression problem given the observation data.

Note that unlike SVDFeature or SVD++ the model allows to take into account additional interaction factors, e.g. it allows to include a "within-class" influence – an influence of entities and aspects of the same type on each other within a single observation. Indeed, indices i, j in $z_i z_j$ term may belong to 2 different items or 2 different features describing the same item. Depending on the problem, such extra interactions can be meaningless or undesirable. In order to control which interactions are allowed in the model one can replace $z_i z_j$ with $\delta_{ij} z_i z_j$, where binary variable δ_{ij} would indicate whether the corresponding interaction is allowed. Clearly, FM can be reduced to any of the previously discussed models by a proper choice of the model parameters and indicator coefficients. A popular variation of FM that uses this technique to separate the latent space for various groups of features is called Field-Aware FM (FFM) [84].

FM models also have a close connection to a higher order approach based on Pairwise Interaction Tensor Factorization (PITF) model [134]. Unlike matrix-based models, PITF uses an array with 3 dimensions, called a 3rd order tensor, to encode pairwise relations between users, items and additional interaction aspects (tags). The model uses 2 independent latent feature spaces for tags: one for user-tag and another one for item-tag relations respectively. The PITF model per se is a member of a broader family of tensor-based methods, which allow to model *n-ary* relations (ternary, quaternary, etc.) not only in a pairwise but in a mutual way.

The topic of tensor methods in recommender systems deserves a separate discussion and we refer the reader to [54] for a comprehensive overview. Worth noting here that tensor-based methods are often used for *context-aware* recommender systems. There are also several direct extensions of the FM idea to higher order cases, e.g. Tensor Machines [188], Higher Order FM [20], Exponential Machines [115].

2.3.5 Remark on connection with SVD

As can be seen, there are some matrix factorization methods that have SVD acronym in their names. This may lead to a certain confusion, that should be avoided. Strictly speaking, most of these methods, like FunkSVD, SVD++, SVDFeature and their derivatives have very little in common with a mathematical formulation of SVD. Un-

like conventional SVD, these methods do not build a space of singular vectors and do not compute singular values. Most of them do not preserve the orthogonality property. Weighted matrix factorization approach is designed specifically to work with incomplete matrices, often ignoring unknown entries or treating them not in the same way as in PureSVD. They form a separate family of methods with different optimization objectives and more flexible tuning. However, due to historical reasons, they are still sometimes are referenced as SVD-based methods.

As a matter of fact, it is, of course, possible to orthogonalize latent factors in Eq. (2.1) and get an equivalent to Eq. (2.5) form with orthonormal basis. This can be achieved by the virtue of the QR decomposition applied to both P and Q matrices (in order to get singular values as well one would have to additionally apply SVD to the product of the low dimensional upper triangular matrices resulted from the QR decomposition). Nevertheless, whenever the optimization objective Eq. (2.17) includes specific constraints other than simple quadratic regularization and the loss function is considerably different from Eq. (2.14), performing orthogonalization potentially leads to a loss of structure in the latent feature space imposed by those special conditions.

Also note that a simple regularization constraint similar to Eq. (2.18) can be added for SVD factors as well. Optimization of the corresponding loss function defined by Eq. (2.4) with this added constraint can be performed without the need to switch to general matrix factorization framework. The solution to such optimization problem, known as *quadratically regularized PCA*, has the same analytical form as the standard SVD and preserves its properties [176]. There is also a connection of the latter to an iterative SVD-based approach called *softImpute* suitable for the rating prediction task [72].

2.4 Learning to rank

There is an overwhelming amount of factorization models that implement sophisticated modifications to standard MF formulation in order to achieve a certain goal or address a specific problem. To name a few, factorization models may include the concept of metric learning [81] or impose additional locality constraints [31] to generate a better latent representation; rely on a more flexible probabilistic inference techniques [105, 143]; use kernel methods to capture non-linear effects [133], etc. An

overview of such a variety of methods falls beyond the scope of this work. We, however, find it necessary to briefly describe a particular example which is directly related to the top-n recommendation problem.

One of the main concerns with the standard formulation of matrix factorization problems is that it is especially suitable for the rating prediction task, however, one can argue that this may not be the best choice for top-n recommendations (see Sec. 1.2.4), where the correct ranking of recommended items is more important than any particular prediction score. It turns out that there is a formal way to address this issue with the help of the *learning to rank* approach [101]. In order to do that, let us consider three general categories of optimization objectives, which lead to different ranking mechanisms in recommender systems: *pointwise*, *pairwise* and *listwise* [28].

Pointwise objective directly depends on a pointwise loss function between the observations and the predicted values. This is the simplest case, which corresponds to previously discussed optimization problems, e.g. Eq. (2.4) or Eq. (2.14), and is not designed for the ranking task. Nevertheless, like in the case with PureSVD, which is formulated as a matrix completion problem and yet can be tuned to provide reasonably good precision-recall scores, it is also possible to empirically find a set of model hyperparameters, which improve the ranking of recommendations. However, it is unlikely to get a significant improvement in this case.

Pairwise objective depends on a pairwise comparison of the predicted values and penalizes those cases where their ordering does not correspond to the ordering of observations. The total loss in that case may take the following (or similar) form:

$$\mathscr{L}(A, R) = \sum_i \sum_{j,j':a_{ij}>a_{ij'}} l(r_{ij} - r_{ij'}),$$

where $l(x_1 - x_2)$ is a pairwise loss function that decreases with the increase of the difference $x_1 - x_2$ (e.g. sigmoid function) and $r_i j$ is the predicted score. It allows to smoothly approximate an indicator function $\mathbb{I}(x_1 > x_2)$.

One of the most popular examples of the pairwise optimization is Bayesian Personalized Ranking (BPR) technique [135], which optimizes a smooth version of AUC with the help of SGD. Another variation of the pairwise approach is Weighted Approximate-Rank Pairwise (WARP) [183, 80], which implements an efficient iterative sampling procedure for negative examples. Alternatively, the authors of RankALS

[172] propose a modification of the WRMF model for the pairwise objective and propose an ALS-based optimization procedure.

Listwise objective optimizes the predicted ordering over entire lists of items at once. The corresponding listwise loss function can be schematically expressed as $l(\{a_{ij}\}, \{r_{ij}\})$. It penalizes the deviation of the predicted ranking of a given list of items from the ground truth ranking based on observations. This approach is considered to be the most suitable for the top-n recommendation task as it allows to directly optimize listwise metrics, e.g. mean average precision (MAP), normalized discounted cumulative gain (NDCG) or mean reciprocal rank (MRR). The listwise approach follows a similar trick of a smooth approximation of the ranking metrics. For example, the reciprocal rank RR_{ij} of an item j recommended to a user i can be approximated by:

$$\mathrm{RR}_{ij} \approx \frac{1}{1 + e^{-r_{ij}}}.$$

A few remarkable examples of this approach are CoFiRank [181], which implements a convex upper bound approximation of NDCG, and CLiMF [156], which instead optimizes a lower bound of a smooth reciprocal rank.

Worth noting here, although both pairwise and listwise algorithms are likely to improve the quality of predicted ranking of elements, they are typically harder to implement and may require additional heuristics to reduce the computational complexity [157].

2.5 Practical aspects

There are many practical aspects that make particular algorithms more suitable in certain environments depending on the desired balance between technical and business requirements. For example, achieving the highest quality of recommendations with a state-of-the-art method may require a lot of computational resources or depend on a complex setup which is hard to maintain and support in production. In such cases a simpler approach with a more straightforward configuration and flexible tuning may become more favorable and help to find a better trade-off between a solution's complexity and the recommendations quality. The latter point is especially crucial when latent factors are used to build neighborhood-based models. In large

Table 2.1: Comparison of low-rank approximation algorithms for explicit feedback data.

Algorithm	Overall complexity	Update complexity	Sensitivity	Optimality
SVD*	$O\left(nnz_A \cdot r + (M+N)r^2\right)$	$O\left(nnz_a \cdot r\right)$	Stable	Global
ALS	$O\left(nnz_A \cdot r^2 + (M+N)r^3\right)$	$O\left(nnz_a \cdot r + r^3\right)$	Stable	Local
CD	$O\left(nnz_A \cdot r\right)$	$O\left(nnz_a \cdot r\right)$	Stable	Local
SGD	$O\left(nnz_A \cdot r\right)$	$O\left(nnz_a \cdot r\right)$	Sensitive	Local

* For both standard and randomized implementations [71].

scale setting an exact search of neighbors may take a prohibitively long time and has to be replaced with approximate solutions (see [10]).

A thorough technical analysis of different algorithms is a non-trivial task and depends on various aspects. One of the most crucial ones is the scalability question, which includes an overall time complexity, memory and storage requirements, online updates support, parallelization efficiency in shared- and distributed-memory environments. Other aspects include stability of an algorithm and its convergence guarantees. General differences between the main algorithms discussed in this chapter are provided in Table 2.1. Note that unlike ALS and SVD, standard implementations of SGD and CD are inapplicable for OCCF problems, as their complexity becomes proportional to the total size of the rating matrix.

2.5.1 Parallel implementations

From the parallelization viewpoint, multi-core shared-memory systems are typically more preferred than distributed shared-nothing environments with multiple computational nodes. This allows to avoid the between-node communication and system state synchronization costs induced by hardware I/O capabilities and specific software implementations.

Moreover, a wide class of large-scale problems can be tackled in the shared-memory settings with the help of an up-to-date hardware [79] and appropriate data preprocessing (e.g. cleansing, subsampling). For example, modern cloud computing services provide instances for memory-intensive applications with *several terabytes* of physical memory onboard, which may help to cover the needs in many practical

cases. Therefore, as a rule of thumb, *distributed setups should be avoided* unless the data and/or model parameters do not fit into a single machine's memory [7].

Parallel SGD

As has been noted, the SGD algorithm is inherently sequential. Model's parameters are updated after every single learning step and parallelization of the algorithm becomes a challenging task. Within such computational environment various architectural choices on the data and model sharing, on data-accessing and data-passing strategies may have a dramatic impact on the algorithm's performance [195, 145].

A straightforward implementation of SGD in shared-memory environment directly leads to overwriting conflicts when, for example, several parallel workers operate on the ratings of the same user/item and, therefore, modify the same vector of user/item latent features. As a result, some of the standard techniques for SGD parallelization, such as *Hogwild!* [130], may not be suitable for the matrix factorization case, unless the data is extremely sparse. As a lock-free algorithm, Hogwild! does not restrict parallel overwrites and it's performance is highly influenced by the data imbalance. Latent features of very popular items or very active users are likely to be updated and recomputed more frequently than latent features of entities with fewer ratings. In practice, this may result in a slower convergence and a degradation of an overall performance of the approach.

In turn, in the distributed case the synchronization of updated parameters of SGD between computational nodes may easily become the main bottleneck of computations. The described problems has led to many different approaches, which achieve certain trade-off's between effective communications and state synchronization, use various data partitioning techniques, implement elaborate locking strategies and rely on aggressive caching. Some of the approaches are only suitable for shared-memory systems [202, 118], others are designed for distributed systems [62, 149] and some support both regimes [192].

Note that in production environments the stochastic nature of SGD may prevent a normal execution of some standard operations. One of the examples is a (non)-regression testing, which is executed on a regular basis to ensure that the behavior of a system during its lifecycle is predictable and stable. An asynchronous execution of SGD in this case may suffer from an uncontrollable randomization at the operating

system level and render an unreproducible state, which makes it harder to track down potential sources of issues and to resolve inconsistencies. In this sense, SVD with its deterministic output and global guarantees provides a reliable alternative.

Parallel ALS

In contrast to SGD, parallel implementation of the standard ALS algorithm for weighted matrix factorization is much more straightforward as latent feature vectors for any user or item can be updated independently at each epoch. It is often said that the algorithm is *embarrassingly parallel*. However, at large scale and in the distributed settings the task may become more involved and IO intensive [190], requiring elaborate data partitioning and parameters' synchronization [39, 150].

As an example, if factor matrices are too large to fit into a single machine's memory than one have to distribute rows of factor matrices P and Q across nodes and properly coordinate the between-node communications to ensure a consistent global state. The interaction data may also be distributed so that all interactions related to a single item or to a single user belong to the same computational node [201]. As this would require switching between columns and rows of the ratings matrix, which is typically stored in the compressed sparse row/column formats (CSR/CSC), two distributed copies of data are used: a column-wise copy for item-related interactions and a row-wise copy for user-related interactions. This allows to avoid redundant computations and reduce the intensity of data transfer.

Nevertheless, communication overhead of the ALS in that case can still be considerable due to random access to the latent feature matrices and may not play well with widely accepted distributed data processing paradigms, such as map-reduce [79]. Alternatively, in the shared-memory settings both ALS and iALS can be implemented very efficiently [60].

Parallel CD

The CD method can be considered as an attempt to combine the advantages of both ALS and SGD methods. It performs alternating optimization similar to ALS and consists of a more lightweight iteration steps. The authors of cyclic CD approach

(CCD++) [190] demonstrate that the method can be relatively easy adapted for both multi-core and distributed environments.

Similarly to [15, 35], in CCD++ the standard row-wise updating scheme is replaced with the column-wise scheme, where the same component of the latent space is updated for all users or items at once. Basically, the CCD++ approach transforms the optimization problem into a sequence of local rank-1 subproblems, where the columns of latent feature matrices are alternatively updated and each alternating step is distributed across several parallel workers. The authors also note that repeating several alternating update cycles within a single subproblem allows to achieve better results.

Implementation of the algorithm is straightforward in the shared-memory settings. In distributed environment it requires additional synchronization of column factors after a complete rank-1 update. However, the authors estimate an overall communication overhead of the approach to be not significantly larger than in the case of popular distributed SGD algorithm [62]. Moreover, it provides a more stable convergence. As demonstrated by the authors the approach shows favorable performance and scales well in both distributed and shared-memory environments.

Parallel SVD

As has been previously discussed, computation of SVD relies on the Lanczos procedure, which requires only matrix-vector products and can be made very efficient with the help of broadly available linear algebra kernels, such as Intel MKL or ARPACK. Internally, the computations are performed by calling highly optimized BLAS/LAPACK routines, tuned for a better utilization of hardware capabilities on the shared memory devices.

Implementation of the algorithm in the distributed setup is typically achieved via the distribution of the Gram matrix-vector product (assuming A is a tall matrix):

$$(A^T A)\boldsymbol{v} = \sum_i \boldsymbol{a}_i(\boldsymbol{a}_i{}^T \boldsymbol{v}).$$

After gathering the result one can use a linear algebra kernel locally to compute the top leading right singular vectors V by the virtue of an eigendecomposition of the Gram matrix. After that the matrix-matrix product AV can also be obtained in a distributed manner. The result is then collected and fed into the standard SVD to

finally get the leading left singular vectors U and the corresponding singular values Σ. The main bottlenecks of the process are parallel data reads and communication overheads incurred by the matrix products. As has been demonstrated in [63] the scaling is very sensitive to implementation details of the distributed calculations and requires a careful investigation to achieve a better scalability.

One of the ways to achieve a better performance in both shared-memory and distributed setups is to replace exact SVD with its approximate variant, such as *Randomized SVD* [71]. A higher computational efficiency of the algorithm comes at the cost of a less accurate result. This, however, is not a stopper as the exact rating prediction is not the main focus of the majority of recommender systems. Moreover, the quality of approximation can be improved by a higher rank.

2.5.2 Hyper-parameters tuning

Due to differences in convergence properties, the methods discussed above require various levels of involvement during the model selection process. Some of the methods are less demanding with respect to the hyper-parameters' choice, others exhibit more sensitive behavior (see Sensitivity column in Table 2.1).

Apparently, SVD can be treated as the most convenient method in this regard. Indeed, it only requires to tune a single parameter – the rank of the decomposition [38]. Moreover, due to optimality of the algorithm, once the PureSVD model is computed for some rank value r, one can immediately obtain a model of any rank $r' < r$ simply by truncating the factor matrices to the first r' components and without any extra computations.

Both CD and ALS depend on at least one extra parameter related to regularization. However, the choice of its value in some reasonable range does not significantly affect the quality of the resulting model and the initialization may play a more important role due to potential abundance of local minima. Nevertheless, WRMF methods introduce additional parameters related to the weighting scheme and require a careful tuning.

Lastly, SGD-based methods are the most sensitive to both initialization and hyper-parameters tuning. This is especially true for the learning rate [202] and many practical implementations employ additional adaptive techniques [22] to automati-

cally select a more appropriate value depending on the convergence pattern and the distance from a minimum.

2.6 Conclusion

This chapter gives an overview of the most popular and widely spread matrix factorization techniques used in collaborative filtering models. It provides the key concepts related to the problem formulation, learning methods, tuning of models and their practical applications. Due to outstanding composability of the MF approach, it allows to address a high number of problems and challenges, arising in recommender systems, that go far beyond simple rating prediction task.

MF methods allow to naturally incorporate additional sources of information and impose specific constraints on the latent feature space, offering more meaningful interpretations and a better quality of recommendations. The algorithms offer various trade-offs between simplicity, computational efficiency, flexibility in tuning, online scenarios support, and quality of recommendations. This remains up to a practitioner to validate the choice of a particular model based on a domain of application, available infrastructure, and business requirements.

Overall, the field of matrix factorization methods in recommender systems has advanced significantly in recent decades. An increasing complexity of problems, especially in hybrid and learning to rank models, has led to the dominance of approximate optimization methods based on ALS, CD, and SGD. These techniques have become a versatile instrument containing a flexible framework for optimization objective manipulation. On the other hand, there are certain practical issues with this approach and, as we argue in Chap. 4, it does not always pay-off in terms of the quality of recommendations.

This naturally raises the question, whether it is possible to employ a more efficient algorithm, like the one used in PureSVD with all its advantages, for solving more complex problems and without giving up on recommendations quality. It turns out that the answer is affirmative. Even though SVD has a very rigid formulation and allows to solve a very specific optimization problem, in Chap. 6 we demonstrate, how it can be tweaked in order to construct an efficient hybrid model, which uses side information to saturate collaborative data and learn more viable behavioral patterns.

Chapter 3

Tensor Factorization

Conventional RS deal with two major types of entities which are typically users (e.g., customers, consumers) and items (e.g., products, resources). Users interact with items by viewing or purchasing them, assigning ratings, leaving text reviews, placing likes or dislikes, etc. These interactions, also called events or transactions, create an observation history, typically collected in a form of transaction/event log that reflects the relations between users and items. Recognizing and *learning* these relations in order to predict new possible interactions is one of the key goals of RS.

As we will see further, the definition of entities is not limited to users and items only. Entities can be practically of any type as long as predicting new interactions between them may bring valuable knowledge and help to make better decisions. In some cases, entities can be even of the same type, like in the task of predicting new connections between people in a social network or recommending relevant paper citations for a scientific paper.

Modern recommender models may also have to deal with more than two types of entities within a single system. For instance, users may want to assign tags (e.g., keywords) to the items they like. Tags become the third type of entity that relates to both users and items, as it represents the user motivation and clarifies items relevance (more on that in Sec. 3.2.2). Time can be another example of an additional entity, as both user preferences and items relevance may depend on time (see Sec. 3.2.3). Taking into account these multiple relations between several entities typically helps to provide more relevant, dynamic and situational recommendations. It also increases the complexity of RS models, which in turn brings new challenges and opens the door for new types of algorithms, such as tensor factorization (TF) methods.

The topic of building a production-ready recommender system is very broad and includes not only algorithms but also concerns a lot about business logic, dataflow design, integration with infrastructure, service delivery and user experience. This also may require specific domain knowledge and always needs a comprehensive evaluation. Speaking about the latter, the most appropriate way of assessing RS quality is an online A/B testing and massive user studies [74, 49, 89], which are typically not available right at hand in academia. In this work, we will only touch mathematical and algorithmic aspects which will be accompanied with examples from various application domains.

3.1 Introduction to tensors

In this section we briefly introduce some general concepts needed for better understanding of further material. For a deeper introduction to the key mathematical aspects of multilinear algebra and tensor factorizations we refer the reader to [90, 37, 68]. As in the case of MF in RS, TF produces a predictive model by revealing patterns from the data. The major advantage of a tensor-based approach is the ability to take into account a multifaceted nature of user-item interactions.

3.1.1 Definitions and notations

We will regard an array of numbers with more than 2 dimensions as a *tensor*. This is a natural extension of matrices to a higher order case. A tensor with m distinct dimensions or *modes* is called an m-way tensor or a tensor of order m.

Without loss of generality and for the sake of simplicity we will start our considerations with a 3rd order tensors to illustrate some important concepts. We will denote tensors with calligraphic capital letters, e.g. $\mathcal{A} \in \mathbb{R}^{M \times N \times K}$ stands for a 3rd order tensor of real numbers with dimensions of sizes M, N, K. We will also use a compact form $\mathcal{A} = [a_{ijk}]_{i,j,k=1}^{M,N,K}$, where a_{ijk} is an element or entry at position (i, j, k), and will assume everywhere in the text the values of the tensor to be real.

Tensor fibers. A generalization of matrix rows and columns to a higher order case is called a *fiber*. Fiber represents a sequence of elements along a fixed mode when all

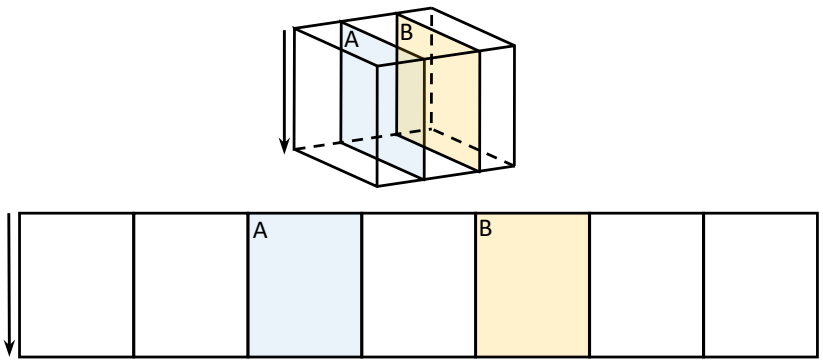

Figure 3.1: Tensor of order 3 (top) and its matricization/unfolding (bottom). Arrow denotes the mode of matricization.

but one indices are fixed. Thus, a mode-1 fiber of a tensor is equivalent to a matrix column, a mode-2 fiber of a tensor corresponds to a matrix row. A mode-3 fiber in a tensor is also called a tube.

Tensor slices. Another important concept is a tensor *slice*. Slices can be obtained by fixing all but two indices in a tensor, thus forming a two-dimensional array, i.e. matrix. In a third order tensor there could be 3 types of slices: horizontal, lateral, and frontal, which are denoted as $\mathcal{A}_{i::}, \mathcal{A}_{:j:}, \mathcal{A}_{::k}$ respectively.

Matricization. Matricization is a key term in tensor factorization techniques. This is a procedure of reshaping a tensor into a matrix. Sometimes it is also called unfolding or flattening. We will follow the definition introduced in [90, Section 2.4]. The n-mode matricization of a tensor $\mathcal{A} \in \mathbb{R}^{M \times N \times K}$ arranges the mode-n fibers to be the columns of the resulting matrix (see Fig. 3.1). For the 1-mode matricization $A_{(1)}$ the resulting matrix size is $M \times (NK)$, for the 2-mode matricization $A_{(2)}$ the size is $N \times (MK)$ and the 3-mode matricization $A_{(3)}$ has the size $K \times (MN)$. In the general case of an m-th order tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_m}$ the n-mode matricization $A_{(n)}$ will have the size $I_n \times (I_1 I_2 \ldots I_{n-1} I_{n+1} \ldots I_m)$. For the corresponding index mapping rules we refer the reader to [90].

Diagonal tensors. Another helpful concept is a diagonal tensor. Tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_m}$ is called diagonal when $a_{i_1 i_2 \ldots i_m} \neq 0$ only if $i_1 = i_2 = \ldots = i_m$. This concept helps to build a connection between different kinds of tensor decompositions.

3.1.2 Problem formulation

As in the matrix case (Sec. 2.1), the main goal is to learn a latent factor model from real observations, which turns into a dimensionality reduction problem with a similar loss function, given by:

$$\mathscr{L}\left(T\left(\mathcal{A}\right), \mathcal{R}(\Theta)\right),$$

where $\mathcal{R}$ is a low rank tensor approximation that may have several distinct forms depending on the type of decomposition used to calculate it (see Sec. 3.1.3); $T(\cdot)$ denotes a problem-dependent data transformation as in the matrix case and Θ stands for model parameters. We will keep this notation throughout the text, i.e. $\mathcal{A}$ will always be used to denote observation data and $\mathcal{R}$ will always be used to represent the reconstructed model, learned from $\mathcal{A}$.

3.1.3 Tensor Factorization techniques

In order to draw a connection to the matrix case and prepare the ground for further generalization to higher order cases we start by introducing an alternative MF notation, which is convenient for our purposes, albeit typically unused. Let us rewrite Eq. (2.5) in the following form:

$$R = \Sigma \times_1 U \times_2 V, \tag{3.1}$$

where $\times_n$ is an *n-mode product*, which is typically defined for the product of a tensor with a matrix. Evidently, in the case of two conformable matrices A and B it has the following form:

$$(A \times_1 B)_{ij} = \sum_k a_{ki} b_{jk}, \quad (A \times_2 B)_{ij} = \sum_k a_{ik} b_{jk}.$$

Expanding this notation to a more general case with some tensor $\mathcal{A}$ gives a conventional definition of an n-mode product [90, Section 2.5]:

$$(\mathcal{A} \times_n B)_{i_1 \ldots i_{n-1} j i_{n+1} \ldots i_m} = \sum_{i_n} a_{i_1 i_2 \ldots i_m} b_{j i_n}. \tag{3.2}$$

For the same purpose of further generalization we also rewrite Eq. (3.1) in two alternative forms, namely, the index form

$$r_{ij} = \sum_{\alpha=1}^{r} \sigma_\alpha u_{i\alpha} v_{j\alpha},$$

and the sum of rank-1 terms

$$R = \sum_{\alpha=1}^{r} \sigma_\alpha \boldsymbol{u}_\alpha \otimes \boldsymbol{v}_\alpha, \tag{3.3}$$

where $\boldsymbol{u}_\alpha, \boldsymbol{v}_\alpha$ denote columns of the factor matrices, e.g. $U = [\boldsymbol{u}_1 \dots \boldsymbol{u}_r], V = [\boldsymbol{v}_1 \dots \boldsymbol{v}_r]$, and $\otimes$ denotes the vector outer product (or dyadic product). Depending on the way these two forms are transformed into a higher order representation, one can arrive at either CANDECOMP/PARAFAC (CP) or Tucker decomposition (TD).

CP decomposition

The most straightforward way of extending SVD to higher orders is to add new factors in Eq. (3.3). In the third order case this will have the following form:

$$\mathcal{R} = \sum_{\alpha=1}^{r} \lambda_\alpha \boldsymbol{u}_\alpha \otimes \boldsymbol{v}_\alpha \otimes \boldsymbol{w}_\alpha, \tag{3.4}$$

where each summation component $\boldsymbol{u}_\alpha \otimes \boldsymbol{v}_\alpha \otimes \boldsymbol{w}_\alpha$ is a *rank-1* tensor. We can also equivalently rewrite Eq. (3.4) in a more concise notation:

$$\mathcal{R} = [[\boldsymbol{\lambda}; U, V, W]], \tag{3.5}$$

where $\boldsymbol{\lambda}$ is a vector of length r with elements $\lambda_1 \geq \dots \geq \lambda_r > 0$ and $U \in \mathbb{R}^{M \times r}$, $V \in \mathbb{R}^{N \times r}$, $W \in \mathbb{R}^{K \times r}$ defined similarly to Eq. (3.3). The expression assumes that factors U, V, W are normalized. As we will see further, in some cases values of $\boldsymbol{\lambda}$ can have a meaningful interpretation. However, in general, the assumption can be safely omitted, which yields:

$$\mathcal{R} = [[U, V, W]] \equiv \sum_{\alpha=1}^{r} \boldsymbol{u}_\alpha \otimes \boldsymbol{v}_\alpha \otimes \boldsymbol{w}_\alpha, \tag{3.6}$$

or in the index from:

$$r_{ijk} = \sum_{\alpha=1}^{r} u_{i\alpha} v_{j\alpha} w_{k\alpha}. \tag{3.7}$$

The right-hand side of Eq. (3.6) gives a rank-r approximation of real observations in the form of CP decomposition. Despite being similar to Eq. (3.3) formulation, there is a number of substantial differences in the concepts of tensor rank and low-rank approximation, thoroughly explained in [90].

Apart from technical considerations, an important conceptual difference is that *there is no higher order extension of the Eckart-Young theorem* (mentioned in Sec. 2.2), i.e. if an exact low-rank decomposition of $\mathcal{A}$ with rank r is known, then its truncation to the first $r' < r$ terms may not give the best rank-r' approximation. Moreover, the optimization task in terms of low-rank approximation is ill-posed [41] which is likely to lead to numerical instabilities and issues with convergence, unless additional constraints on factor matrices (columns orthogonality, non-negativity, etc.) are imposed.

Tucker decomposition

A stable way of extending SVD to the higher order case is to transform the diagonal matrix Σ from Eq. (3.1) into a third order tensor $\mathcal{G}$ and add an additional mode-3 tensor product, defined by Eq. (3.2), with a new factor matrix W:

$$\mathcal{R} = [[\mathcal{G}; U, V, W]] \equiv \mathcal{G} \times_1 U \times_2 V \times_3 W, \tag{3.8}$$

where $U \in \mathbb{R}^{M \times r_1}, V \in \mathbb{R}^{N \times r_2}, W \in \mathbb{R}^{K \times r_3}$ are typically required to be *columnwise orthonormal* and have a similar meaning of the latent feature matrices as in the case of SVD. Tensor $\mathcal{G} \in \mathbb{R}^{r_1 \times r_2 \times r_3}$ is called a *core tensor* of the TD and a tuple of numbers (r_1, r_2, r_3) is called a multilinear rank. The index form of TD reads:

$$r_{ijk} = \sum_{\alpha,\beta,\gamma=1}^{r_1,r_2,r_3} g_{\alpha\beta\gamma}\, u_{i\alpha}\, v_{j\beta}\, w_{k\gamma}. \tag{3.9}$$

The decomposition is not unique; however, the optimization problem with respect to multilinear rank is *well-posed*. Also note that setting the core tensor $\mathcal{G}$ to be diagonal turns the decomposition into CP form.

The definition of TD is not restricted to have 3 modes only. Generally, the number of modes is not limited; however, storage requirements depend exponentially on the number of dimensions (see Table 3.1), which is often referred as a *curse of dimensionality*. This imposes strict limitations on the number of modes for many practical cases, whenever more than 4 entities are modelled in a multilinear way (e.g. *user*, *item*,

	CP	TD	TT	HT
storage	dnr	$dnr + r^d$	dnr^2	$dnr + dr^3$

Table 3.1: Storage requirements for different TF methods. For the sake of simplicity, this assumes a tensor with d dimensions of equal size n and all ranks (or rank in case of CP) of a tensor decomposition set to r.

time, *location*, *company* or any other context variables, see Fig. 1.1). In order to break the curse of dimensionality, a number of efficient methods has been developed recently, namely Tensor Train (TT) [116] and Hierarchical Tucker (HT) [67]. However, we are not aware of any published results related to TT- or HT-based implementations in RS.

3.1.4 Optimization algorithms

TF techniques rely on the same concepts of *pointwise*, *pairwise* and *listwise* optimization, introduced in Sec. 2.4. An optimization problem can be written in a similar form, keeping in mind the difference in model parameters, i.e.,

$$\Theta^* = \arg\min_{\Theta} \mathscr{J}(\Theta), \tag{3.10}$$

where $\Theta := \{U, V, W\}$ for CP-based models and $\Theta := \{\mathcal{G}, U, V, W\}$ in the case of TD. The difference in factorization techniques also leads to variations in how exactly the solution to Eq. (3.10) is obtained. Below we provide some typical examples.

Pointwise algorithms for TD

In case of TD-based model the solution to Eq. (3.10) in a standard least squares sense can be found with help of two well-known methods proposed in [40]: Higher-Order SVD (HOSVD) [163, 168, 127] or Higher-Order Orthogonal Iteration (HOOI) [197, 165].

The HOSVD method can be described as a consecutive application of SVD to all 3 matricizations of $\mathcal{A}$, i.e. $A_{(1)}, A_{(2)}, A_{(3)}$ (assuming that missing data is imputed with zeros). Generally it produces a suboptimal solution; however, it is worse than the best possible solution only by a factor of $\sqrt{d}$, where d is the number of dimensions [70]. Due to its simplicity this method is often used in recommender systems literature.

Algorithm 1: Practical HOOI algorithm for Tucker decomposition

Input : Tensor $\mathcal{A}$ in sparse COO format,
Tensor decomposition ranks r_1, r_2, r_3

Output: $\mathcal{G}, U, V, W$

Initialize V and W by random matrices with orthonormal columns.

repeat

$U \leftarrow r_1$ leading left singular vectors of $A^{(1)}(W \otimes V)$

$V \leftarrow r_2$ leading left singular vectors of $A^{(2)}(W \otimes U)$

$W, \Sigma, Z \leftarrow r_3$ leading singular triplets of $A^{(3)}(V \otimes U)$

$\mathcal{G} \leftarrow$ reshape matrix ΣZ^T into shape (r_3, r_1, r_2) and transpose

until *norm of the core ceases to grow or exceeds maximum iterations*;

The HOOI method uses an iterative procedure based on an alternating least squares (ALS) technique, which successively optimizes the objective with the help of SVD (see Alg. (1)). In practice it may require a small amount of iterations to converge to an optimal solution, but in general it is not guaranteed to find a global optimum [90]. The choice of any of these two methods for particular problem may require additional investigation in terms of both computational efficiency and recommendations quality before the final decision is made.

The orthogonality constraints imposed by TD may in some cases have no specific interpretation. Relaxing these constraints leads to a different optimization scheme, typically based on gradient methods, such as stochastic gradient descent (SGD) [85]. The objective in that case is expanded with a regularization term $\Omega(\Theta)$:

$$\mathscr{J}(\Theta) = \mathscr{L}(T(\mathcal{A}), \mathcal{R}(\Theta)) + \Omega(\Theta), \tag{3.11}$$

which is commonly expressed as follows:

$$\Omega(\Theta) = \lambda_G \|\mathcal{G}\|_F^2 + \lambda_U \|U\|_F^2 + \lambda_V \|V\|_F^2 + \lambda_W \|W\|_F^2, \tag{3.12}$$

where $\lambda_G, \lambda_U, \lambda_V, \lambda_W$ are regularization parameters and usually $\lambda_U = \lambda_V = \lambda_W$.

Pointwise algorithms for CP

As has been noted in Sec. 3.1.3, CP is generally ill-posed and if no specific domain knowledge could be employed to impose additional constraints, a common ap-

proach to alleviate the problem is to introduce regularization similarly to Eq. (3.12):

$$\Omega(\Theta) = \lambda_U \|U\|_F^2 + \lambda_V \|V\|_F^2 + \lambda_W \|W\|_F^2, \tag{3.13}$$

Indeed, depending on the problem formulation it may also have more complex form both for CP (e.g. as in Sec. 3.2.3) and TD models. In general, regularization allows to ensure convergence and avoid degeneracy (e.g. when rank-1 terms become close to each other by absolute value but their magnitudes go to infinity and have opposite signs [90]); however, it may lead to a sluggish rate of convergence [110]. In practice, however, many problems can still be solved with CP using variations of both ALS [78, 91] and gradient-based methods.

Pairwise and listwise algorithms.

Pairwise and listwise methods are considered to be more advanced and accurate as they are specifically designed to solve ranking problems. The objective function is often derived directly from a definition of some ranking measure, e.g. pairwise AUC or listwise MAP (see [137] for CP-based and [157] for TD-based implementations), or constructed in a way that is closely related to those measures [134, 138].

These methods typically have a non-trivial loss function with complex data interconections within it which makes it hard to optimize and tune. In practice, the complexity problem is often resolved with help of handcrafted heuristics and problem-specific constraints (see Sec. 3.2.2 and Sec. 3.2.4), which simplify the model and improve computational performance.

3.2 Tensor-based models in recommender systems

Treating data as tensor may bring new levels of flexibility and/or quality into RS models; however, there are nuances that should be taken into account and treated properly. This section covers different tensorization techniques used to build advanced RS in various application domains. For all the examples we will use a unified notation (where it is possible) introduced in Sec. 3.1, hence it might look different from the notation used in the original papers. This helps to reuse some concepts within different models and build a consistent narrative throughout the text.

3.2.1 Personalized search and resource recommendations

There is a very tight connection between personalized search and RS. Essentially, recommendations can be considered as a *zero query search* [6] and, in turn, personalized search engine can be regarded as a query-based RS.

Personalized search systems aim at providing a better search experience by returning the most relevant results, typically web pages (or resources), in response to a user's request. A clicktrough data (i.e. an event log of clicks on the search results after submitting a search query) can be used for this purpose as it contains an information about users' actions and may provide valuable insights into search patterns. The essential part of this data is not just a web page that a user clicks on, but also a context, a query associated with every search request that carries a justification for the user's choice. The utility function in that case can be formulated as:

$$f_u : \text{User} \times \text{Resource} \times \text{Query} \rightarrow \text{Relevance Score},$$

where Resource denotes a set of web pages and Query is a set of keywords that can be specified by users in order to emphasize their current interests or information needs. In the simplest case a single query can consist of one or a few words (e.g. "jaguar" or "big cat"). More elaborate models could employ additional natural language processing tools in order to breakdown queries into a set of single keywords, e.g. a simple phrase "what are the colors of the rainbow" could be transformed into a set {"rainbow", "color"} and further split into 2 separate queries, associated with the same (*user, resource*) pair.

CubeSVD

One of the earliest and at the same time very illustrative works where this formulation was explored with help of tensor factorization is CubeSVD [163]. The authors build a 3-rd order tensor $\mathcal{A} \in \mathbb{R}^{M \times N \times K}$ with values representing the level of association (the relevance score) between user i and web-page j in the presence of query k:

$$\begin{cases} a_{ijk} > 0, & \text{if } (i,j,k) \in S, \\ a_{ijk} = 0, & \text{otherwise,} \end{cases}$$

where S is an observation history, e.g. a sequence of events described by the triplets (*user, resource, query*). Note that authors in their work use simple queries without processing, e.g. "big cat" is a single query term.

The association level can be expressed in various ways, the simplest one is to measure a co-occurrence frequency f, e.g. how many times a user has clicked on a specific page after submitting a certain query. In order to prevent an unfair bias towards the pages with high click rates, it can be restricted to have only values of 0 (no interactions) or 1 (at least one interaction). Or it can be rescaled with a logarithmic function:

$$f' = \log_2(1 + f/f_0),$$

where f' is a new normalized frequency and f_0 is, for example, an IDF (Inverse Document Frequency) measure of a web page. Another scaling approach can also be used.

The authors proposed to model the data with a third order TD Eq. (3.8) and in order to find it they applied the HOSVD. Similarly to SVD Eq. (2.5), factors $U \in \mathbb{R}^{M \times r_1}, V \in \mathbb{R}^{N \times r_2}$ and $W \in \mathbb{R}^{K \times r_3}$ represent embedding of users, web pages and queries vectors into a lower-dimensional latent factors space with dimensionalities r_1, r_2 and r_3 correspondingly. The core tensor $\mathcal{G} \in \mathbb{R}^{r_1 \times r_2 \times r_3}$ defines the form and the strength of multilinear relations between all three entities in the latent feature space. Once the decomposition is found, the relevance score for any (*user, resource, query*) triplet can be recovered with Eq. (3.9).

With the introduction of new dimensions the data sparsity becomes even higher, which may lead to a numerical instabilities and general failure of the learning algorithm. In order to mitigate that problem, the authors propose several smoothing techniques: based on value imputation with small constant and based on the content similarity of web pages. They reported an improvement in the overall quality of the model after these modifications.

After applying the decomposition technique the reconstructed tensor $\mathcal{R}$ will contain new non-zero values denoting potential associations between users and web resources influenced by certain queries. The tensor values can be directly used to rank a list of the most relevant resources: the higher the value r_{ijk} is the higher the relevance of the page j to the user i within the query k.

This simple TF model does not contain a remedy for some of the typical RS problems such as cold start or real-time recommendations and is most likely to have

issues with scalability. Nevertheless, this work is very illustrative and demonstrates the general concepts for building a tensor-based RS.

TOPHITS

As has been discussed in Sec. 1.2.6, new entities can appear in the system dynamically and rapidly, which in the case of higher order models creates even more computational load, i.e. full recomputation of tensor decomposition quickly becomes infeasible and incremental techniques should be used instead. However, in some cases simply redefining the model might lower the complexity. As we mentioned in Sec. 1.1.2, a simple approach to reduce the model is to eliminate one of the entities with some sort of aggregation.

For example, instead of considering (*user, resource, query*) triplets we could work with aggregated (*resource, resource, query*) triplets, where every frontal slice $\mathcal{A}_{::k}$ of the tensor is simply an adjacency matrix of a resources browsed together under a specific query. Therefore users are no longer explicitly stored and their actions are recorded only in the form of a co-occurrence of resources they searched for.

An example of such a technique is TOPHITS model [91]. This analogy requires an extra explanation as the authors are not modelling users clicking behavior. The model is designed for web-link analysis of a static set of web pages referencing each other via hyperlinks. The data is collected by crawling those web pages and collecting not only links but also keywords associated with them. However, the crawler can be interpreted as a set of users browsing those sites by clicking on the hyperlinked keywords. This draws the connection between CubeSVD and TOPHITS model as the keywords can be interpreted as a short search queries in that case. And, as we stated earlier, users (or crawlers) can be eliminated from the model by constructing an adjacency matrix of linked resources.

The authors of the TOPHITS model extend an adjacency matrix of interlinked web pages with additional keyword information and build the so called adjacency tensor $\mathcal{A} \in \mathbb{R}^{N \times N \times K}$ that encodes hubs, authorities and keywords. As has been mentioned, the keyword information is conceptually very similar to queries, hence it can be also modelled in a multirelational way. Instead of TD format the authors prefer to use CP in the form of Eq. (3.5) with $U, V \in \mathbb{R}^{N \times r}$ and $W \in \mathbb{R}^{K \times r}$ with ALS-based optimization.

The interpretation of this decomposition is different from the CubeSVD. As the authors demonstrate, the weights λ_k, $(1 \leq k \leq r)$ have a straightforward semantic meaning as they correspond to a set of r specific topics extracted from the overall web page collection. Accordingly, every triplet of vectors (u_k, v_k, w_k) represents a collection of hubs, authorities and keyword terms respectively, characterized by a topic k. The elements with higher values in these vectors provide the best-matching candidates under the selected topic, which allows a better grouping of web pages within every topic and provide means for a personalization.

For example, as the authors show, a personalized ranked list of authorities $\boldsymbol{a}^*$ can be obtained with:

$$\boldsymbol{a}^* = V \Lambda W^T \boldsymbol{q}, \tag{3.14}$$

where $\Lambda = \mathrm{diag}(\boldsymbol{\lambda})$ is a diagonal matrix and $\boldsymbol{q}$ is a user-defined query vector of length K with elements $q_t = 1$ if term t belongs to the query and 0 otherwise, $t = \{1, \ldots, K\}$. Similarly, a personalized list of hubs can be built simply by substituting factor V with U in Eq. (3.14).

The interpretation of tensor values might seem very natural; however, there is an important note to keep in mind. Generally, the restored tensor values might turn both positive and negative, and in most applications the negative values have no meaningful explanation. The non-negative tensor factorization (NTF) [36, 200] can be employed to resolve that issue (see example in [32], and also the connection of NTF to probabilistic factorization model under a specific conditions [33]).

3.2.2 Social tagging

A remarkable amount of research is devoted to one specific domain, namely social tagging systems (STS), where predictions and recommendations are based on commonalities in social tagging behavior (also referred as collaborative tagging). A comprehensive overview of the general challenges and state-of-the-art RS methods can be found in [104].

Tags carry a complementary semantic information that helps STS users to categorize and organize items of their choice. This brings an additional level of interpretation of the user-item interactions, as it exposes the motives behind the user preferences and explains the relevance of particular items. This observation suggests that

tags play an important role in defining the relevance of (*user, item*) pairs, hence all the three entities should be modelled mutually in a multirelational way. The scoring function in that case can be defined as follows:

$$f_u : \text{User} \times \text{Item} \times \text{Tag} \rightarrow \text{Relevance Score}.$$

The triplets (*user, item, tag*), coming from an observation history S, can be easily translated into a 3rd order tensor $\mathcal{A} = [a_{ijk}]_{i,j,k=1}^{M,N,K}$, where M, N, K denote the number of users, items and tags respectively. Users are typically not allowed to assign the same tags to the same items more than once, hence the tensor values are strictly binary and defined as:

$$\begin{cases} a_{ijk} = 1, & \text{if } (i,j,k) \in S, \\ a_{ijk} = 0, & \text{otherwise.} \end{cases}$$

Unified framework

As in the case of keywords and queries, tensor dimensionality reduction helps to uncover latent semantic structure of the ternary relations. The values of the reconstructed tensor $\mathcal{A}$ can be interpreted as the likeliness or weight of new links between users, items and tags. These links might be used for building recommendations in various ways: help users assign relevant tags for items [169], find interesting new items [170], or even find like-minded users [167].

The model that is built on top of all three possibilities is described in [168]. The authors perform a latent semantic analysis on the data with help of the HOSVD. Generally, the base model is similar to CubeSVD (see Sec. 3.2.1): items can be treated as resources and tags as queries.

The authors also face the same problem with sparsity. The tensor matricizations $A_{(n)}, 1 \leq n \leq 3$ within the HOSVD procedure produce highly sparse matrices which may prevent the algorithm from learning the accurate model. In order to overcome that problem they propose a smoothing technique based on a *kernel trick*.

In order to deal with the problem of real-time recommendations (see Sec. 1.2.6) the authors adopt a well known *folding-in method* [56] to a higher order case. The folding-in procedure helps to quickly embed a previously unseen entity into the latent features space without recomputing the whole model. For example, an update to a

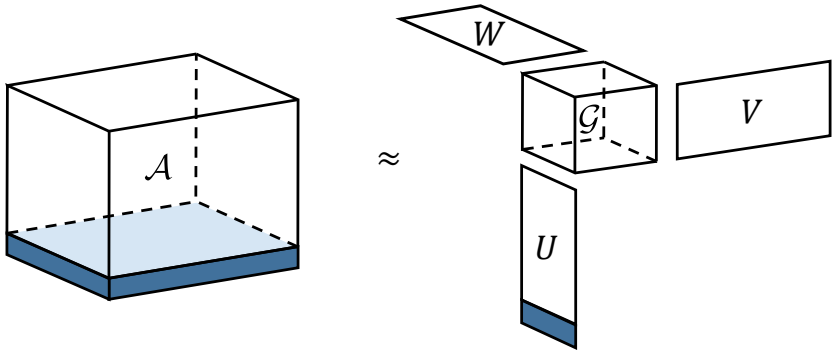

Figure 3.2: Higher order folding-in for Tucker decomposition. A slice with new user information in the original data (left) and a corresponding row update of the factor matrix in TD (right) are marked with solid color.

users feature matrix U can be obtained with:

$$\boldsymbol{u}_{new} = \boldsymbol{p} V_1 \Sigma_1^{-1},$$

where $\boldsymbol{p}$ is a new user information that corresponds to a *row* in the matricitized tensor $A_{(1)}$; V_1 is an already computed (during HOSVD step) matrix of right singular vectors of $R_{(1)}$, and Σ_1 is a corresponding diagonal matrix of singular values; $\boldsymbol{u}_{new}$ is an *update row* which is appended to the latent factor matrix U. The resulting update to reconstructed tensor $\mathcal{R}_{new}$ is computed with (see Fig. 3.2):

$$\mathcal{R}_{new} = [\mathcal{G} \times_2 V \times_3 W] \times_1 \begin{bmatrix} U \\ \boldsymbol{u}_{new} \end{bmatrix},$$

where the term within the left brackets of the right hand side does not contain any new values, e.g. does not require the full recomputation and can be pre-stored, which makes the update procedure much more efficient.

Nevertheless, this typically leads to a loss of orthogonality in factors and negatively impacts the accuracy of the model in the long run. This can be avoided with an *incremental SVD update*, which for the matrices with missing entries was initially proposed by [23]. As the authors demonstrate, it can be also adopted for tensors.

It should be noted that this is not the only possible option for incremental updates. For example, a different incremental TD-based model with HOOI-based optimization is proposed in [197] for a highly dynamic, evolving environment (not related to tag-based recommendations). The authors of this work use an extension of a two-dimensional incremental approach from [139].

RTF and PITF

The models, overviewed so far, has a common "1/0" interpretation scheme for a missing values, i.e. all triplets $(i, j, k) \in S$ are assumed to be positive feedback and all others (missing) are negative feedback with zero relevance score. However, as the authors of ranking with TF model (RTF) [137] and more elaborate pairwise interaction TF (PITF) [134] model emphasize, all missing entries can be split into 2 groups: the true negative feedback and the unknown values. The true negatives correspond to those triplets of (i, j, k) where the user i has interacted with the item j and has assigned tags different from the tag k. More formally, if P_S is a set of all posts that correspond to all observed (*user, item*) interactions, than true negative feedback within any interaction between user i and item j is defined as:

$$A_{ij}^{-} := \{k | (i, j) \in P_S \wedge (i, j, k) \notin S\}.$$

Likewise, true positive feedback is:

$$A_{ij}^{+} := \{k | (i, j) \in P_S \wedge (i, j, k) \in S\}.$$

All other entries are unknowns and are to be uncovered by the model.

Furthermore, both RTF and PITF models do not require any specific values to be imposed on either known or unknown entries. Instead they only impose pairwise ranking constraints on the reconstructed tensor values:

$$r_{ijk_1} > r_{ijk_2} \Leftrightarrow (i, j, k_1) \in A_{ij}^{+} \wedge (i, j, k_2) \in A_{ij}^{-}.$$

These post-based ranking constraints become the essential part of an optimization procedure. The RTF model uses the Tucker format; however, it aims at directly maximizing AUC measure, which, according to the authors, takes the following form:

$$AUC(\Theta, i, j) := \frac{1}{|A_{ij}^{+}||A_{ij}^{-}|} \sum_{k^{+} \in A_{ij}^{+}} \sum_{k^{-} \in A_{ij}^{-}} \sigma(r_{ijk^{+}} - r_{ijk^{-}}),$$

where Θ are the parameters of the TD model (as defined in Eq. (3.11)) and $\sigma(x)$ is a sigmoid function, introduced to make the term differentiable:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{3.15}$$

As we are interested in maximizing AUC, the loss function takes the form:

$$\mathscr{L}(T(\mathcal{A}),\, \mathcal{R}) = -\sum_{(i,j)\in P_S} AUC(\Theta, i, j).$$

The regularization term of the model is defined by Eq. (3.13).

The authors adopt a stochastic gradient descent (SGD) algorithm for solving the optimization task. However, as they state, directly optimizing the AUC objective is computationally infeasible. Instead, they exploit a smart trick of recombining and reusing precomputed summation terms within the objective and its derivatives. They use this trick for both tasks of learning and building recommendations.

The PITF model is built on top of ideas from RTF model. It adopts Bayesian Personalized Ranking (BPR) technique proposed for MF case in [135] to the ranking approach. The tags rankings for every observed post (i, j) are not deterministically learned like in RTF model but instead are derived from the observations by optimizing the maximum aposteriori estimation. This leads to a similar to RTF optimization objective with similar regularization (excluding the tensor core term which is not present in CP) and slightly different loss function:

$$\mathscr{L}(T(\mathcal{A}),\, \mathcal{R}) = -\sum_{(i,j,k_1,k_2)\in D_S} \ln \sigma(r_{ijk_1} - r_{ijk_2}),$$

where the same notation as in RTF is used; σ is a sigmoid function from Eq. (3.15) and D_S is a training data, i.e. a set of quadruples:

$$D_S = \{(i, j, k_1, k_2) \,|\, (i, j, k_1) \in S \wedge (i, j, k_2) \notin S\}. \tag{3.16}$$

An important difference of PITF from RTF is that the complexity of multilinear relations is significantly reduced by leaving only pairwise interactions between all entities. From the mathematical viewpoint it can be considered as a CP model with a special form of partially fixed factor matrices (cf. Eq. (3.7)):

$$r_{ijk} = \sum_{\alpha} u_{i\alpha}\, w^U_{k\alpha} + \sum_{\alpha} v_{j\alpha}\, w^V_{k\alpha} + \sum_{\alpha} u_{i\alpha}\, v_{j\alpha}, \tag{3.17}$$

where $w^U_{k\alpha}$ and $w^V_{k\alpha}$ are the parts of the same matrix W responsible for tags relation to users and items respectively; $u_{i\alpha}$ and $v_{j\alpha}$ are interactional parts of U and V.

The authors emphasize that the user-item interaction term does not contribute to the BPR-based ranking optimization scheme which yields even more simple equation that becomes an essential part of the PITF model:

$$r_{ijk} = \sum_{\alpha} u_{i\alpha} w^{U}_{k\alpha} + \sum_{\alpha} v_{j\alpha} w^{V}_{k\alpha}. \tag{3.18}$$

Another computational trick that helps to train the model even faster without sacrificing the quality is random sampling within the SGD routine. All the quadruples in D_S corresponding to a post (i, j) are highly overlapped with respect to the tags associated with them. Therefore, learning with some randomly sampled quadruples is likely to have a positive effect on learning the remaining ones.

In order to verify the correctness and effectiveness of such simplifications the authors conduct experiments with both BPR-tuned TD and CP and demonstrate that PITF algorithm achieves close or even better quality of recommendations while learning features faster than the other two TF methods.

Despite its computational effectiveness, the original PITF model is lacking the support for the real-time recommendation scenarios, where rebuilding the full model for each new user, item or tag could be prohibitive. The authors of [99] overcome this limitation by introducing the folding-in procedure compatible with the PITF model and demonstrate its ability to provide high recommendations quality. Worth noting here that a number of variations of the folding-in technique are available for different TF methods, see [196].

The idea of modelling higher order relations in a joint pairwise manner similar to Eq. (3.18) has been explored in various application domains and is implemented in various settings, either straightforwardly or as a part of a more elaborate RS model [57, 75, 199, 152]. There are several generalized models [182, 138], [78], which also use this idea. They are covered in more details in Sections Sec. 3.2.4 and Sec. 3.2.4 of this work.

Improving the prediction quality

As has been already mentioned in Sec. 3.2.2 high data sparsity typically leads to a less accurate predictive models. This problem is common across various RS domains. Another problem, specific to STS, is tag ambiguity and redundancy. The following are

the examples of some of the most common techniques, developed to deal with these problems.

The authors of CubeRec [187] propose a clustering-based separation mechanism. This mechanism builds clusters of triplets (*user, item, tag*) based on the proximity of tags derived from the item-tag matrix. With this clustering some of the items and tags can belong to several clusters at the same time, according to their meaning. After that the initial problem is split into a number of sub-problems (corresponding to clusters) of a smaller size and hence, with a more dense data. Every subproblem is then factorized with the HOSVD similarly to [163], and the resulting model is constructed as a combination of all the smaller TF models.

The authors of the clustering-based TD model (ClustHOSVD) [165] also employ clustering approach. However, instead of splitting the problem, they replace tags by tag clusters and apply the HOOI method (which is named AlsHOSVD by the authors) directly to the modified data consisting of (*user, item, tag cluster*) triplets. They also demonstrate the effect of different clustering techniques on the quality of RS.

As can be seen, many models benefit from clustering either prior to or after the factorization step. This suggests that it can also be beneficial to perform simultaneous clustering and factorization. This idea is explored by the authors of [55], where they demonstrate the effectiveness of such an approach.

A further improvement can be achieved with hybrid models (see Sec. 1.1.2), which exploit a content information and incorporate it into a tensor-based CF model. It should be noted, however, that there is no "single-bullet" approach, suitable for all kinds of problems, as it highly depends on the type of data used as a source of content information.

The authors of [109] exploit acoustic features for music recommendations in a tag-based environment. The features, extracted with specific audio-processing techniques, are used to measure the similarity between different music samples. The authors make an assumption that similarly sounding music is likely to have similar tags, which allows to propagate tags to the music that was not tagged yet. With this assumption the data is augmented with new triplets of (*user, item, tag*), which leads to a more dense data and results in a better predictive quality of the HOSVD model.

The TF and tag clustering (TFC) model [127] combines both content exploitation and tag clustering techniques. The authors focus on the image recommendations

problem, thus they use an image processing techniques in order to find items' similarities and propagate highly relevant tags. Once the tag propagation is completed, the authors find tag clusters (naming them topics) and build new association triplets (*user, item, topic*), which are further factorized with the HOSVD.

As a last remark in this section, the idea of model splitting, proposed in the CubeRec model, was also explored in a more general setup in [179]. The authors consider a multiple context environment, where user-item interactions may depend on various contexts such as location, time, activity, etc. This is generally modelled with an N-th order tensor, where $N > 3$. Instead of dealing with higher number of dimensions and greater sparsity, the authors propose to build a separate model for every context type, which transforms the initial problem into a collection of a smaller problems of order 3. Then all the resulting TF models are combined with specific weights (based on the context influence measure proposed by the authors) and can be used to produce recommendations. However, despite the ability to better handle the sparsity issue, the model may loose some valuable information about the relations between different types of context. A more general methods for multi-context problems are covered in Sec. 3.2.4.

3.2.3 Temporal models

User consumption patterns may change in time. For example, the interest of TV users may correlate not only with a topic of a TV program, but also with a specific time of the day. In retail user preferences may vary depending on the season. Temporal models are designed to learn those time-evolving patterns in data by taking the time aspect into account, which can be formalized with the following way scoring function:

$$f_u : \text{User} \times \text{Item} \times \text{Time} \rightarrow \text{Relevance Score}.$$

Even though the general problem statement looks already familiar, when working with the Time domain one should mind the difference between the evolving and periodic (e.g. seasonal) events which may require a special treatment.

BPTF

One of the models that exploits periodicity of events is the Bayesian Probabilistic TF (BPTF) [186]. It uses seasonality to reveal trends in retail data and predict the orders that will arrive in the ongoing season based on the season's start and previous purchasing history. The key feature of the model is the ability to produce forecasts on the sales of the new products that were not present in previous seasons. The model captures dynamic changes in both product designs and customers' preferences solely from the history of transactions and does not require any kind of an expert knowledge.

The authors develop a probabilistic latent factors model by introducing priors on the parameters; i.e. the latent feature vectors are allowed to vary and the variance of relevance scores is assumed to follow a Gaussian distribution:

$$r_{ijk}|U, V, W \sim \mathcal{N}(< U_{i:}, V_{j:}, W_{k:} >, \gamma^{-1}),$$

where γ is an observations precision and $< U_{i:}, V_{j:}, W_{k:} >$ denotes a right hand side of Eq. (3.7). Note that in the original work the authors use a transposed version of the factor matrices, e.g. any column of the factor U in their work represents a single user, the same holds for two other factors.

In order to prevent the overfitting the authors also impose prior distributions on U and V:

$$U_{i:} \sim \mathcal{N}(0, \sigma_U^2 I),$$
$$V_{j:} \sim \mathcal{N}(0, \sigma_V^2 I).$$

Furthermore, the formulation for the time factor W takes into account its evolving nature and implies smooth changes in time:

$$W_{k:} \sim \mathcal{N}(W_{k-1:}, \sigma_{dW}^2 I),$$
$$W_{0:} \sim \mathcal{N}(\mu_W, \sigma_0^2 I).$$

The time factor W rescales the user-item relevance score with respect to the time-evolving trends and the probabilistic formulation helps to account for the users who do not follow those trends.

The authors show that maximizing the log-posterior distribution $\log p(U, V, W, W_{0,:}|\mathcal{A})$ with respect to U, V, W and $W_{0:}$ is equivalent to an optimization task with the weighted square loss function:

$$\mathscr{L}(T(\mathcal{A}), \mathcal{R}) = \sum_{(i,j,k)\in S} (a_{ijk} - r_{ijk})^2, \tag{3.19}$$

and a bit more complex regularization term:

$$\Omega(\Theta) = \frac{\lambda_U}{2}\|U\|_F^2 + \frac{\lambda_V}{2}\|V\|_F^2 + \frac{\lambda_{dT}}{2}\sum_{k=1}^{K}\|W_{k:} - W_{k-1:}\|^2 + \frac{\lambda_0}{2}\|W_{0:} - \mu_W\|^2,$$

where $\lambda_U = (\alpha\sigma_U)^{-1}, \lambda_V = (\alpha\sigma_V)^{-1}, \lambda_{dW} = (\alpha\sigma_{dW})^{-1}, \lambda_0 = (\alpha\sigma_0)^{-1}$ and the last two terms are due to a dynamic problem formulation. The number of parameters of this model makes the task of optimization almost infeasible. However, the authors come up with an elaborate MCMC-based integration approach that makes the model almost parameter-free and also scales well.

TCC

The authors of TF-based subspace clustering and preferences consolidation model (TCC) [180] exploit the periodicity in usage patterns of the IPTV users in order to, at first, identify them and, secondly, provide with more relevant recommendations, even if those users share the same IPTV account (for example, across all family members). This gives a slightly different definition of a utility function:

$$f_u : \text{Account} \times \text{Item} \times \text{Time} \rightarrow \text{Relevance Score},$$

where Account is the domain of all registered accounts and the number of accounts is not greater than the number of users, i.e. $|\text{Account}| \leq |\text{User}|$. Initial tensor $\mathcal{A}$ is built from the triplets (*account, item, time*) and its values are just the play counts.

In order to be able to find a correct mapping of the real users to the known accounts, the authors introduce a concept of a *virtual user*. Within the model the real user is assumed to be a composition of particular virtual users u_{ak} which express the specific user's preferences tied to a certain time periods, e.g.:

$$u_{ak} := \{(a, p_k) \,|\, a \in A, p_k \in P, p_k \neq \varnothing\},$$

where a is an account from the set of all accounts A, p_k is a sub-period from the set of all non-overlapping time periods P.

As the authors state, manually splitting the time data into the time slots p_k does not fit the data well and they propose to find those sub-periods from the model. They first solve the SGD-based optimization task for TD with the same weighted squared loss function as in Eq. (3.19) and regularization term as in Eq. (3.12) (with $\lambda_G = \lambda_U = \lambda_V = \lambda_W = \frac{1}{2}\lambda$). Once the model factors are found, the sub-periods p_k can be obtained by clustering the time feature vectors:

$$P \leftarrow \text{ k-Means clustering of the rows of } W.$$

Then the consolidation of virtual users into the real ones can be done in 2 steps. At first, a binary similarity measure is computed between different pairs of virtual users $(u_{ak}, u_{ak'})$ corresponding to the same account a. The second step is to combine similar virtual users so that every real user is represented as a set of virtual ones. This is done with help of a graph-based technique. Once the real users are identified, recommendations can be produced with a user-based kNN approach. As the authors demonstrate, the proposed method not only provides a tool for user identification, but also outperforms standard kNN and TF-based methods applied without any prior clustering.

3.2.4 General context-aware models

In previous sections we have discussed TF methods targeted at specific classes of problems: keyword- or tag-based recommendations, temporal models. They all have one thing in common - the use of a third entity leading to a higher level of granularity and better predictive capabilities of a model. This leads to an idea of generalization of such approach, which is suitable for any model formulated in the form of Eq. (1.2).

Multiverse

One of the first attempts towards this generalization is the Multiverse model [85]. The authors define context as any set of variables that influence users' preferences and propose to model it by the N-th order TD with $N-2$ contextual dimensions:

$$\mathcal{A} = [[\mathcal{G}; U, V, W_1, W_2, \ldots, W_{N-2}]],$$

where factors $W_i,\ i \in \{1,\ldots,N-2\}$ represent a corresponding embedding of every contextual variable into a reduced latent space and all factors including U and V are not restricted to be orthogonal. As the authors state, the model is suitable for any contextual variables over a finite categorical domain. It should be noted that the main focus of the work is systems with an explicit feedback and the model is optimized for the error-based metrics, which does not guarantee an optimal items ranking as has been discussed in Sec. 1.2.4.

Following the general form of an optimization objective stated in Eq. (3.11), the authors use the weighted loss function:

$$\mathscr{L}\left(T(\mathcal{A}),\ \mathcal{R}\right) = \frac{1}{\|\mathcal{G}\|_1} \sum_{(i,j,k)\in S} l(a_{ijk},\ r_{ijk}),$$

where $l(a_{ijk},\ r_{ijk})$ is a pointwise loss function, which can be based on l_1, l_2 or other types of distance measures. The example is provided for the 3rd order case; however, it can be easily generalized to a higher orders. The authors also use the same form of the regularization term as in Eq. (3.12), as it enables trivial optimization procedure.

In order to fight against the growing complexity for the higher order cases they propose a modification of the SGD algorithm. Within a single optimization step the updates are performed on every row of the latent factors independently. For example, an update for i-th row of U:

$$U_{i:} \leftarrow U_{i:} - \eta\,\lambda_U U_{i:} - \eta\,\partial_{r_{ijk}} l(a_{ijk},\ r_{ijk})$$

is independent on all other factors and thus all the updates can be performed in parallel. The parameter η defines the model's learning step.

In addition to the general results on the real dataset, this work features a comprehensive experimentation on the semi-synthetic data that shows the impact of a contextual information on the RS models performance. It demonstrates that high context influence leads to a better quality of the selected context-aware methods, among which the proposed TF approach gives the best results, while a context-unaware method's quality significantly degrades.

TFMAP

Similarly to the previously discussed PITF model, the TF for MAP optimization model (TFMAP) [157] also targets optimal ranking; however, it exploits the MAP

metric instead of AUC. The model is designed for an implicit feedback systems which means that the original tensor $\mathcal{A}$ is binary with non-zero elements reflecting the fact that an interaction has occurred:

$$a_{ijk} = \begin{cases} 1, & \text{if } (i,j,k) \in S, \\ 0, & \text{otherwise.} \end{cases} \tag{3.20}$$

The optimization objective is drawn from the MAP definition:

$$MAP = \frac{1}{MK} \sum_{i=1}^{M} \sum_{k=1}^{K} \frac{\sum_{j=1}^{N} \frac{a_{ijk}}{p_{ijk}} \sum_{j'=1}^{N} a_{ij'k} \mathbb{I}\left(p_{ij'k} \leq p_{ijk}\right)}{\sum_{j=1}^{N} a_{ijk}},$$

where p_{ijk} denotes the position (or rank) of item j in the item list of user i under the context of type k; $\mathbb{I}(\cdot)$ is an indicator function, which is equal to 1 if the condition is satisfied and 0 otherwise, both depend on the reconstructed values of $\mathcal{A}$. In order to make the metric smooth and differentiable the authors propose two approximations:

$$\frac{1}{p_{ijk}} \approx \sigma(r_{ijk}),$$
$$\mathbb{I}\left(p_{ij'k} \leq p_{ijk}\right) \approx \sigma(r_{ij'k} - r_{ijk}),$$

where r_{ijk} is calculated with Eq. (3.7) (which makes the model a CP-based) and σ is a sigmoid function defined by Eq. (3.15). Notably, $r_{ij'k} - r_{ijk} =< U_{i:}, V_{j':} - V_{j:}, W_{k:} >$, where we use the same notation as in BPTF model, see Sec. 3.2.3. The model also follows the standard optimization formulation stated in Eq. (3.11), where the loss function is just a negative MAP gain, i.e.

$$\mathscr{L}(T(\mathcal{A}), \mathcal{R}) = -MAP,$$

and the regularization has the form of Eq. (3.13).

Note that MAP optimization also has a weighted form due to Eq. (3.20). However, the computation complexity would still be prohibitively high due to its complex structure. In order to mitigate that, the authors propose the fast learning algorithm: for each (*user, context*) pair only a limited set of a representative items (a buffer) is considered, which in turn, allows to control the computational complexity. They also provide an efficient algorithm of sampling the "right" items and constructing the buffer, which does not harm the overall quality of the model.

CARTD

The CARTD model (Context-Aware Recommendation Tensor Decomposition) [182, 138] provides a generalized framework for an arbitrary number of contexts and also targets an optimal ranking instead of a rating prediction. Under the hood the model extends the BPR-based ranking approach used in the PITF model to the higher order cases.

The authors introduce a unified notion of an entity. A formal task is to find the list of the most relevant entities within a given contextual situation. Remarkably, all information that is used to make recommendations more accurate and relevant, is defined as a context. In that sense, not only information like tag, time, location, user attributes, etc. is considered to be a context, even users themselves might be defined as a context of an item. This gives a more universal formulation for the recommendations task:

$$f_u : \text{Entity} \times \text{Context}_1 \times \ldots \times \text{Context}_n \rightarrow \text{Relevance Score}. \tag{3.21}$$

As an illustration to that, a quadruple (*user, item, time, location*) maps to ($context_1$, *entity,* $context_2$, $context_3$). Obviously, the definition of the entity depends on the task. For example, in case of social interactions prediction with (*user, user, attribute*) triplets, the main entity as well as one of the context variables will be a user.

The observation data in a typical case of a user-item interactions can be encoded similarly to Eq. (3.16):

$$D_S := \{(e, f, c_1, \ldots, c_n) \mid (e, c_1, \ldots, c_n) \in S \wedge (f, c_1, \ldots, c_n) \notin S\},$$

where e and f are the entities (i.e. items) and $c_i, i = \{1, \ldots, n\}$ denotes a context type (includes users). As the authors emphasize, this leads to a huge sparsity problem, and instead they propose to relax conditions and instead build the following set for learning the model:

$$D_A := \left\{(e, f, c_1, \ldots, c_n) \mid \forall c_i : \#_{c_i}(e) > \#_{c_i}(f)\right\},$$

where $\#_{c_i}(\cdot)$ indicates the number of occurrences of an entity within the context c_i. The rule $\#_{c_i}(e) > \#_{c_i}(f)$ denotes the prevalence of the entity e over the entity f with respect to all possible contexts.

The optimization objective will also look similar to the one used in the PITF model with the loss function defined as:

$$\mathscr{L}\left(T(\mathcal{A}),\, \mathcal{R}\right) = - \sum_{(e,f,c_1,\ldots,c_n)\in D_A} \ln \sigma\left(r_{\{c\},e} - r_{\{c\},f}\right),$$

where $\{c\}$ denotes a set of all context variables c_i and the tensor values a_{ijk} are calculated with help of the reduced CP model with the pairwise only interactions, similarly to Eq. (3.18):

$$r_{\{c\},e} = \sum_{i=1}^{n} v_e^{E,C_i} v_{c_i}^{C_i,E},$$

where v_e^{E,C_i} and $v_{c_i}^{C_i,E}$ are the elements at the cross section of the e-th row and the i-th column of the factor matrices V^{E,C_i} and $V^{C_i,E}$ respectively. As in the previous cases, the regularization term $\Omega(\Theta)$ have similar to Eq. (3.13) form, which includes all the factors from Θ:

$$\Theta = \{V^{E,C_1}, V^{C_1,E}, \ldots, V^{E,C_n}, V^{C_n,E}\}.$$

iTALS

As has been mentioned in the introduction (see Sec. 1.2.3), an implicit feedback does not always correlate with the actual user preferences, thus a simple binary scheme (as in Eq. (3.20)) may not be accurate enough. For this reason, the authors of the iTALS model (ALS-based implicit feedback recommender algorithm) [77] propose to use the confidence-based interpretation of an implicit feedback introduced in [82] and adopt it for the higher order case.

They introduce the dense tensor $\mathcal{W}$ that assigns non-zero weights for both observed and unobserved interactions. For the n-th order tensor it has the following form:

$$\begin{cases} w_{i_1,\ldots,i_n} = \alpha \cdot \#(i_1,\ldots,i_n), & \text{if } (i_1,\ldots,i_n) \in S, \\ w_{i_1,\ldots,i_n} = 1, & \text{otherwise,} \end{cases} \tag{3.22}$$

where $\#(i_1,\ldots,i_n)$ is the number of occurrences of the tuple $(i_1,\ldots,i_n)$ (e.g. the combination of the user i_1 and the item i_2 interacted within the set of contexts $i_3,\ldots,i_n$) in the observation history; α is set empirically and $\alpha \cdot \#(i_1,\ldots,i_n) > 1$ which means that the observed events provide more confidence in the user preferences than the unobserved ones.

The loss function will then take the form:

$$\mathscr{L}(T(\mathcal{A}),\, \mathcal{R}) = \sum_{i_1,\ldots,i_n} w_{i_1,\ldots,i_n}(a_{i_1,\ldots,i_n} - r_{i_1,\ldots,i_n})^2,$$

where weights $w_{i_1,\ldots,i_n}$ are defined by Eq. (3.22), $r_{i_1,\ldots,i_n}$ are the values of a binary feedback tensor of order n, defined similarly to Eq. (3.20), and $a_{i_1,\ldots,i_n}$ are the values of the reconstructed tensor.

The model uses CP with an ALS-based optimization procedure and a standard regularization similar to Eq. (3.13). The latent feature vectors are encoded in the rows of the factor matrices, not the columns, i.e. following the authors' notation, we should rewrite Eq. (3.6) as:

$$\mathcal{A} = [[M_1^T, \ldots, M_n^T]],$$

where M_i $(1 \leq i \leq n)$ are transposed factors of the CP decomposition.

The authors show, how an efficient computation over the dense tensor can be achieved with the same tricks that are used in [82] for the matrix case. The model also has a number of modifications [76]: based on the conjugate gradient approach (iTALS-CG) and the coordinates descent approach (iTALS-CD) where an additional features compression is achieved by the Cholesky decomposition. This makes the iTALS-CD model to learn even faster than MF methods. While performing on approximately the same level of accuracy as the state-of-the-art Factorization Machines (FM) method [136], it is capable of learning more complex latent relations structure. Another modification is the pairwise "PITF-like" reduction model, named iTALSx [75].

GFF

The General Factorization Framework (GFF) [78] further develops the main ideas of the family of iTALS models. Within the GFF model different CP-based factorization models (also called a preference models) are combined in order to capture the intrinsic relations between users and items influenced by an arbitrary number of contexts. As in many other works the authors of GFF model fix the broad definition of the context as an entity, which "value is not determined solely by the user or the item", i.e. not a content information (see Sec. 1.2.7).

The model can be better explained with the example. Let us consider the problem of learning the scoring function as follows:

$$f_u : \mathrm{U} \times \mathrm{I} \times \mathrm{S} \times \mathrm{Q} \rightarrow \mathrm{Relevance\,Score}, \tag{3.23}$$

where U and I are the domains of *users* and *items* respectively; S stands for *season* and denotes the periodicity of the events (see Sec. 3.2.3); Q describes the sequential consumption patterns, e.g. what are the previous items that were also consumed with the current one (see [77] for broader set of examples). Let us also define the pairwise interactions between users and items as UI (standard CF model), between items and seasons as IS and so forth. Using the same notation we can also define multi-relational interactions, such as UIS for a 3-way user-item-season interactions or UISQ for the 4-way interactions between all 4 types of entities.

In total, there could be 2047 different combinations of interactions, yet not all of them are feasible in terms RS model, as not all of them may contribute to the preference model.

As the result, GFF generates a very flexible multirelational model that allows to pick the most appropriate scheme of interactions, which does not explode the complexity of the model and meanwhile achieves a high quality of recommendations. Based on the experiments the authors conclude: "leaving out useless interactions results in more accurate models".

As it can be seen, tensors-based methods help represent and model complex environments in a convenient and congruent way, suitable for various problem formulations. More examples, starting from multi-criteria learning to cross-domain recommendations, can be found in [54]. Nevertheless, as we have already stated earlier, the most common practical task for RS is to build a ranked list of recommendations (a top-n recommendations task). In this regard, we summarize related features of some of the most illustrative, in our opinion, methods in Table 3.2. We also note that while, technically, incremental learning is applicable to any of these methods, not all authors provide the specific steps to implement it. We therefore emphasize, whether the presented models explicitly address the problem of real-time recommendations in dynamic environments (the last column of the table).

Table 3.2: Comparison of popular TF models. The *Ranking prediction* column shows whether a method is evaluated against ranking metrics. The *Online* column denotes a support for real-time recommendations for new users (e.g. folding-in).

Name	Type	Algorithm	Domain	Entities	Optimization	Ranking prediction	Online
TOPHITS [91], 2005	CP	ALS	Link prediction	Resources, Keyword	pointwise	✓	
CubeSVD [163], 2005	TD	HOSVD	Personalized Search	User, Resource, Query	pointwise	✓	
RTF [137], 2009	TD	SGD	Folksonomy	User, Item, Tag	pairwise	✓	
BPTF [186], 2010	CP	MCMC	Temporal dynamics	User, Item, Time	pointwise		
Multiverse [85], 2010	TD	SGD	Context-awareness	User, Item, Contexts	pointwise		
PITF [134], 2010	CP*	SGD	Folksonomy	User, Item, Tag	pairwise	✓	
TagTR [168], 2010	TD	HOSVD	Folksonomy	User, Item, Tag	pointwise	✓	✓
TFMAP [157], 2012	CP	SGD	Context-awareness	User, Item, Context	listwise	✓	
CARTD [138], 2012	CP*	SGD	Context-awareness	Item, Contexts	pairwise	✓	
ClustHOSVD [165], 2015	TD	HOOI	Folksonomy	User, Item, Tag	pointwise	✓	
GFF [78], 2015	CP*	ALS	Context-awareness	User, Item, Contexts	pointwise	✓	

* Method uses pairwise reduction concept, initially introduced in PITF.

3.3 Conclusion

In this chapter, we have attempted to overview a broad range of tensor-based methods used in recommender systems to date. As we have seen, these methods provide a powerful set of tools for merging various types of additional information and are aimed at increasing flexibility, customizability and quality of recommendation models. Tensorization enables original and non-trivial setups, going far beyond standard user-item paradigm, and finds its applications in various domains. Tensor-based models can also be used as a part of more elaborate systems, providing compressed latent representations as an input for other well-developed techniques.

One of the main concerns for the higher order models is inevitable growth of computational complexity with increasing number of dimensions. Even for mid-sized production systems that have to deal with highly dynamic environments, this might have negative implications, such as the inability to generate recommendations for new users in a timely manner. This type of issues can be firmly addressed with incremental updates and higher order folding-in techniques.

Despite a numerous amount of various TF techniques, we note a common pattern shared by all of them: user feedback, if present in the form of rating-like values, is always treated as a direct measure of item relevance on a continuous real-valued scale.

On the other hand, as we will discuss in Chap. 4, it seems more natural to treat ratings as ordinal values. In Chap. 5 we explore this idea more carefully with the help of tensor-based formulation, where user feedback is encoded within a third dimension. In some sense, it resembles context-aware and tag-based approaches, like Multiverse or TagTR, if we view it from the ternary relations perspective.

We demonstrate that our model can potentially increase the perceived relevance of recommendations, especially in the warm start scenario, where generated predictions are highly sensitive to user input and, therefore, the ability to build a more adequate preference model plays a crucial role. Our method also supports a simple analytic form of the folding-in calculation, which results from a direct higher order generalization of the one used in PureSVD.

In addition to that, as we will show in Chap. 5, even though the formulation of our tensor approach is pointwise, it can also be viewed as a special kind of a ranking task. Moreover, it has a much simpler formulation, comparing to pairwise or listwise learning to rank methods like PITF or TFMAP; it does not bring additional complexity and does not require any heuristic-based simplifications for solving the problem.

Finally, in Chap. 7 we combine both tensor-based preference model and our hybrid model from Chap. 6 within a new tensor factorization approach. Unlike the methods like Multiverse, it is more economical in a sense that it does not require building a separate latent feature space for incorporating additional side knowledge into the model.

Chapter 4

Limited preference information problem

As we note in Sec. 1.1.2, the general premise of the CF approach is that it does not require any expert or domain knowledge for generating a model. Intrinsic relations within the data are expected to be learned directly from the observed user-item interactions and in the case of latent factor models enclosed in the latent feature space. While it may hold true in many cases, there are still situations when the output of CF models becomes unreliable.

A particular example of that is the problem of the insufficient amount of user feedback used to build CF models and to generate recommendations with their help. We will denote that problem with the term *limited preference information*. It can be divided into two distinct subproblems: the *local lack of preferences* and the *global lack of preferences*, both having different implications on the behavior and the outcome of the models.

The local problem arises right after an entity is introduced to the system for the first time and lasts until a certain amount of preferences is collected. Predictions for the entity during this phase are typically less reliable and less stable, especially in the beginning. While this phase is inevitable for any entity, we implicitly assume here that there are no general obstacles for collecting more data with time, which eventually resolves the problem. In other words, the problem does not affect the quality of recommendations in general; hence the name "local".

In contrast, the main driver for the global problem is the very mechanism of user-item interactions, which limits an overall amount of preferences that can be potentially collected. As it was noted in Sec. 1.2.2, this may have a negative impact on general quality of recommendations. We note, however, that unlike the previous

case, the problem is not always present and significantly depends on the domain of application, the nature of decision making and some other factors.

Below we provide a more detailed view on each subproblem and demonstrate a few "blind spots" in existing approaches, which we *aim to resolve in this work*.

4.1 Local lack of preferences

In the definition of the local problem, we assume that the observed data is generally "nice" in a sense that it contains enough collaborative information for learning representative patterns and building an adequate CF model. This corresponds to the cases of a comparatively high density of user-item interactions. In this setting, the task of generating relevant recommendations becomes problematic only in particular cases of new users or new items with insufficient or unavailable feedback data, i.e., in the warm start and the cold start scenarios introduced in Sec. 1.2.1. We refer to this subproblem as the *local lack of preferences*.

There are several possible ways of dealing with this problem. One way is to use the so-called *rating elicitation* technique. In this case, a recommender system is designed to explicitly ask newly arrived or unrecognized users to provide feedback on a pre-selected non-personalized list of items before they can actually start using a recommendation service. Conversely, in the case of newly introduced items, a predefined group of existing users can be asked to provide some feedback on them.

The selection process, however, is a very challenging task. For example, in the new user case, a trivial strategy of presenting randomly picked items is ineffective and can be frustrating for users. A more adequate approach is to find a collection of the most representative items [53]. However, deriving a list of items that helps to better learn user preferences without making a user feel bored by the elicitation process is a non-trivial problem. In fact, it represents only a small part of a more general research, devoted to the so-called *active learning* approach [51].

The problem becomes even worse if a pre-built list of items resonates poorly with actual user tastes. In this case provided feedback will be mostly negative, i.e., full of items with low scores or low ratings assigned by users. Taking into account *the tendency of conventional CF algorithms to favor positively rated items*, which we demonstrate in Sec. 5.1.1, the elicitation process, in the end, may lead to an undesired

outcome with a lot of irrelevant recommendations. This is often alleviated by generating more items until enough positive feedback (e.g., items with high scores or high ratings) is gathered. However, this makes engagement with the system less effortless for users and may even lead to a loss of interest in it.

As an alternative to the rating elicitation option, one can infer user interests and item relevance based on their side information. For example, one can restrict the list of shown items based on which category of items a user is currently browsing. This is where various content-based models can be applied. However, this approach should be primarily considered as a last resort due to limitations mentioned in Sec. 1.1.1. If side information is broadly available, a more elaborate but more fruitful alternative is to use a hybrid approach. This may not only help to better handle the cold start, but also allows to mitigate the global problem of insufficient preference information described next.

4.2 Global lack of preferences

The aforementioned "nice" structure of data is not something that can be always easily achieved. In practice, data sparsity is often very high, and in the most severe cases, it may prevent CF models from discovering non-trivial and reliable relations within the data, which in turn leads to substantial degradation of the resulting quality of recommendations. We will refer to this subproblem as the *global lack of preferences*.

A trivial example of the global lack of preferences is a fresh system start when there is yet not enough data collected by a recommender system. Clearly, in some cases, the problem can be resolved merely by allowing a recommender system to run for a longer period of time. However, there are more challenging situations, when, for example, due to domain specifics users interact with only a small fraction of all available items and the sparsity problem may even get worse with time. This is often the case in online stores with a large (and potentially increasing) assortment. *The availability of side information plays a crucial role in such cases.*

Indeed, as we have mentioned in Sec. 1.1.1, user choice can be influenced by intrinsic properties of items. For example, users may prefer products of a particular brand or products with specific characteristics. Even if item properties are unknown, the knowledge about users themselves, such as demographic information or occupa-

tion, may help to explain their choice and to reveal certain behavioral patterns. By using some hybrid CF approach, one may try to uncover at least some of these patterns and build a better predictive model. In addition to an improved quality of recommendations, using side information may help to make algorithms more stable and less susceptible to extreme data sparsity.

4.3 Related work

The described subproblems can be addressed with a broad range of methods and techniques. Below we provide a brief overview of such methods with respect to the subproblem they are suitable for.

4.3.1 Addressing the local problem

Let us first consider the rating elicitation process. The majority of rating elicitation systems seem to focus only on a positive experience of users without properly addressing the negative part. Roughly, a common approach across many standard models can be summarized with a short line "people who like this item also like ...". This dictates how the rating elicitation is performed as well: users are asked to provide information about items they like. Obviously, this approach covers at most only half of possible scenarios, as users may also dislike items, presented to them during the rating elicitation phase. At best, the disliked items are simply filtered out. However, it seems more natural to engage with a complementary "people who *dislike* this item do like ... instead" scenario. This, in turn, requires a more careful analysis of user feedback, which can also improve the quality of recommendations in a general scenario.

Nevertheless, only a few research papers have studied an effect of different types of user experience on the quality of recommender systems in general. For example, the authors of [96] proposed to split user ratings into categories and compute the relevance scores based on user-specific rating values distribution. The authors of SLIM method [113] have compared models that learn ratings either explicitly (r-models) or in a binary form (b-models). They compare initial distribution of ratings with that of recommended items by calculating a hit rate for every unique rating value. The

authors show that r-models have a stronger ability to predict top-rated items even if ratings with the highest values are not prevalent.

The authors of [27] state that discovering what a user does not like can be easier than discovering what the user actually likes. They propose to simply ignore all negative preferences of individuals to avoid unsatisfactory recommendations to a group. Meanwhile, excessive focus on high ratings can lead to an even more pronounced bias of recommendation algorithms towards top-rated items [51]. Thus, for example, the authors of [50] explore the effect of items with the lowest predicted score on the outcome of the personalized rating elicitation procedure. Their conclusion, however, is that while it helps to improve performance in terms of error-based metrics, the technique is ineffective for improving the ranking of recommendations. The authors of the MinRating approach [86], used for exploration of items, additionally show that the lowest score predictions, made by MF models, are often inaccurate (leading to higher gradients in parameter updates), which additionally exposes models' biases.

The authors of [8] provide an insightful study of subjective nature of ratings from a user perspective. They demonstrate that a rating scale is non-uniform in a sense that distances between different rating values are perceived differently even by the same user. The work raises an important question of a more adequate preference modeling. As the authors show, the nature of user ratings is very different from its common representation in terms of a simple real-valued scale or as a set of cardinal numbers and is better interpreted in terms of an ordinal concept. The studies in neoclassical economics [103] also support this observation.

An ordinal representation, in turn, can be achieved with a number of methods. For example, one can apply several variants of ordinal regression [191, 95]. Alternatively, it is possible to use various probabilistic frameworks, e.g., Bayesian inference for matrix factorization [119], Boltzmann Machines (BM) [122] or Restricted Boltzmann Machines (RBM) [144]. An improper explicit dependence on the values of feedback can also be alleviated within the learning to rank framework, briefly described in Sec. 2.4, and in particular with the methods like xCLiMF [158] (an extension of CLiMF to explicit feedback data) or CoFiRank [181].

Nevertheless, while all these methods have their own unique set of advantages, they also come with some drawbacks and limitations. For example, the methods based on the BM approach are incredibly versatile and provide a natural formulation for or-

dinal data representation; however, they have high computational complexity and cannot be easily applied to large-scale problems [144, 95]. Moreover, finding an optimal structure of BMs can be quite cumbersome due to the need to tune additional hyper-parameters, including the number of hidden units and connectivity in the input layer.

A more scalable alternative would be OrdRec [95], an ordinal regression-based model. The model additionally allows to have a personalized rating scale, which addresses individual differences in user perception, and at the same time, it does not require any additional latent features to encode rating values. However, its latent space for items is twice the size of that in standard MF models: for every interaction, it encodes an item's neighborhood with a separate set of latent features. In addition to that, while the model is based on the SVD++ approach, it takes more time to train without providing a significant improvement in the quality of recommendations. Thus, it may not always be worth the efforts, especially taking into account that in some cases comparable or even better improvement over SVD++ can be achieved by PureSVD [38, 96] with much fewer efforts spent on its tuning [1].

As we have noted in Sec. 2.4, learning to rank models also employ more elaborate optimization objectives, which may involve special tricks and depend on non-trivial optimization techniques. For example, CoFiRank [181] optimizes the upper bound approximation of nDCG, which additionally invokes a linear program along with the SGD-based optimization. The xCLiMF model [158] in that sense is more advantageous, as it implements a more straightforward SGD-based optimization of a smooth approximation of the Expected Reciprocal Rank. Nevertheless, even though both models are designed to optimize a ranking metric, they may still underperform a fine-tuned SVD++ algorithm in top-n recommendations task [98].

Moreover, as we have discussed in Sec. 2.5.2, any SGD-based approach, including SVD++, requires careful hyper-parameters' tuning, which is far more labor-intensive than tuning an SVD-based model. Indeed, assuming that we have 60 unique points on a hyper-parameter grid, we would have to retrain and evaluate an SGD-based model 60 times. In contrast, the SVD-based model needs to be trained only

[1] Metrics used in [95] and [38, 96] are not precisely the same; however they are still of the same type and are expected to correlate. The difference between them in comparative analysis is unlikely to be as dramatic as the difference between the ranking- or relevance-based metrics on the one hand and the error-based metrics on the other, which was mentioned in Sec. 1.2.4 and also discussed in [38].

once with some sufficiently high rank value, and then all models of a lower rank are obtained simply by truncating the number of columns in factor matrices. Even if an SVD-based model would depend on additional hyper-parameters, still for every fixed set of their values one can quickly evaluate the model for a range of ranks by computing it only for the highest one. This can significantly reduce the time for finding an optimal configuration and favorably distinguishes SVD-based models from other MF models.

In addition to that, while implementing some advanced MF method is typically not too hard, making it computationally efficient can be quite challenging. We have covered a few significant difficulties concerning both shared-memory and distributed environments in Sec. 2.5.1, which require good engineering skills and a certain level of experience in multicore programming and high-performance computing. Inefficient implementations may easily result in days of training time, which can make them infeasible in rapidly changing environments that require more frequent updates of the entire model.

On the other hand, while there are a few production-level implementations, still many advanced algorithms, even if publicly available, are often distributed in the form of an unparalleled code running on a single-core, which requires substantial efforts for making them really efficient. Moreover, new and more elaborate optimization objectives would require writing new optimization code. In that sense, the ability to reuse SVD seems more beneficial, as it already has highly optimized implementations in many programming languages and takes advantage of efficient linear algebra kernels. This minimizes the amount of new code to be written, which in turn saves both time and efforts and also allows to enjoy additional guarantees backed by solid linear algebra.

4.3.2 Addressing the global problem

We have already discussed the problem of sparsity in Sec. 1.2.2 and provided a few examples that allow to address it with some sort of “smoothing” techniques or data clustering. It is also possible to use the compressed sensing technique in CF settings [198].

In this work we are more interested in the cases where additional information about relations is also available. We will specifically focus on content data, i.e., side information about users and items, which allows using the hybrid approach with a number of versatile and powerful techniques. We consider a particular category of hybrid algorithms that are based on the MF approach. For convenience, we informally group them into several categories, depending on a particular choice of data preprocessing steps and optimization objectives.

We have already described a few hybrid models in Sec. 2.3.4, namely FM and SVDFeature. These models represent a wide class of hybrid factorization approaches, where real attributes and properties are mapped to latent features with the help of some *linear transformation*. In the majority of models feature mapping is a part of optimization process [123, 29, 140]; however, it can also be used as a post-processing step [58]. Several other models can be categorized into an *aggregation* approach [114, 185], where feature-based relations are imposed on interaction data and are used for learning aggregated representations. Alternatively, in the *augmentation* approach features are represented as dummy variables that extend the model [108].

Another wide class of models uses regularization-based techniques to enforce proximity of entities in the latent feature space based on their actual features. Some of these models are based on *probabilistic frameworks* [69, 126], others extend standard MF objective [111, 31] with additional terms. One of the most straightforward and well-studied regularization-based approach is *collective MF* [159]. In its simplest variant, sometimes also called *coupled MF*, the corresponding loss function can be formulated as [52]:

$$\mathscr{L}(A,\Theta) = \left\| W \circ \left(A - PQ^T\right)\right\|_{\mathsf{F}}^2 + \alpha\left(\|H - PW_H\|_{\mathsf{F}}^2 + \|G - QW_G\|_{\mathsf{F}}^2\right),$$

where $\Theta = \{P, Q, W_H, W_G\}$ represents model parameters, matrices H and G encode side information, i.e. user attributes and item characteristics, and α controls the contribution of the side information into the resulting model.

There are also several variations of the coupled factorization technique, where parametrization of the content matrices H, G is replaced with parametrization of the content-based similarity between users and items [155, 11]. The authors of the Local Collective Embeddings (LCE) model [147] propose to add a locality constraint, so that the entities, which are close to each other in terms of real features, remain close to

each other in the latent feature space as well. The authors use the model to specifically address the cold-start problem.

Many of the described techniques can be generalized to the tensor case as well. As an example, the coupled tensor factorization approach is explored in [128]. However, as we have shown in Chap. 3, the tensor format provides great flexibility in data representation and problem formulation. For example, one of the most well-known TD-based models called *Multiverse* (Sec. 3.2.4) encodes any data, including side information, within additional dimensions. This model, however, can be hardly applied to the problems with many side features encoded in a separate dimension, as the storage required for TD factors depends exponentially on the number of dimensions. This leads to the *curse of dimensionality* problem, mentioned in Sec. 3.1.3. Moreover, as we have discussed in Sec. 1.2.8, encoding side information within additional dimensions may lead to interpretation issues.

Alternatively, the Contextual Tensor-Based Approach for Personalized Expert Recommendation (TAPER) model [61] uses CP format as a workhorse for a unified representation. Various sources of information in TAPER are glued together with the help of additional regularization constraints. The model also imposes locality constraints, requiring similar entities to remain close to each other in the latent feature space. The curse of dimensionality problem is avoided due to the use of CP format. However, as have been already mentioned in Sec. 3.1.3, the CP decomposition is generally unstable and may require some efforts in order to ensure convergence. Moreover, in general, it does not impose orthogonality constraints on the columns of factor matrices. This leads to a more complicated folding-in procedure that requires additional optimization steps.

Worth noting that the majority of these methods are also based on SGD optimization. This leads to the same technical issues, which were discussed at the end of the previous section.

4.4 The need for new methods

To summarize, the described methods offer great flexibility in solving the problem of insufficient preference information. However, in many cases, it comes at the cost of elaborate optimization schemes that require substantial engineering efforts

for implementing them and a special care on hyper-parameter tuning, which may not necessarily result in significant quality improvement.

Moreover, different methods focus on either the local or the global aspect of the problem. Meanwhile, both subproblems can be present at the same time within a single system. Switching between different methods to address particular aspects of the problem would inevitably increase maintenance costs. In turn, a joint model based on some derivative of these methods is likely to increase an overall complexity of the solution and can potentially magnify the aforementioned technical issues.

On the other hand, in Sec. 2.2 and Sec. 2.5 we have demonstrated that SVD-based methods can be adjusted quite considerably and, in addition to that, offer many advantages in terms of scalability, hyper-parameter tuning and support for online settings. Moreover, the deterministic output and global guarantees, provided by SVD, can play a crucial role in production environments. All this naturally leads to the question, whether it is possible to derive a new approach, which would help to solve the problems described in this chapter and at the same time preserve all the advantages provided by SVD. Ideally, we would like to have a method that satisfies the following criteria:

- uses SVD or applies it sequentially for optimization,
- requires minimal tuning of hyper-parameters,
- supports highly dynamic online settings via folding-in,
- efficiently handles a large number of different side features,
- provides a unified solution to the limited preference information problem and can be easily adapted to each of its subproblems.

In the next part, we will gradually develop a new approach that meets all the aforementioned requirements. In Chap. 5 we firstly address the problem of proper feedback modeling. We propose a tensor-based method, where feedback values are encoded within an additional dimension along with users and items, which allows to naturally impose ordinal relations in the model. Technically it resembles the context-aware methods and can also be viewed as a simplified version of the learning to rank approach. In Chap. 6 we step back to the matrix case in order to see how the PureSVD

model can be modified in order to incorporate side information into it. We propose a generalized formulation, which allows to achieve that while staying within the computational paradigm of SVD. Finally, in Chap. 7 we combine both approaches into a single framework, that preserves the advantages of its both predecessors and provides a unified solution to the problem of limited preference information.

Part II

Proposed models

Chapter 5

Higher order preference model

We argue that the situation with the local lack of preferences, described in the previous chapter, can be alleviated if the system is able to learn equally well from both positive and negative feedback. Consider the following movie recommendation example. A new user marks the "Scarface" movie with a low rating, e.g., 2 stars out of 5, and no other user preferences are available. This is likely to indicate that the user does not like movies about crime and violence. It also seems natural to assume that the user probably prefers "opposite" features, such as sentimental story, which can be present in romantic movies, or happy and joyful narrative provided by animation or comedies. If this is the case, asking to rate or simply recommending the "Godfather" movie is a redundant and inappropriate action. Similarly, if a user provides some negative feedback for the first part of a series (e.g., the first movie from the "Lord of the rings" trilogy), it is quite natural to expect that the system will not immediately recommend another part from the same series.

A more proper way to engage with the user, in that case, is to leverage a sort of *"users, who dislike that item, do like these items instead"* scenario. Users certainly can share preferences not only in what they like but also in what they do not like and it is fair to expect that techniques, based on the CF approach, could exploit this for more accurate predictions even from solely negative feedback. In addition to that, negative feedback may have a greater importance for a user, than a positive one. Some psychological studies demonstrate that emotionally negative experience not only has a stronger impact on an individual's memory [87], but also has a more significant effect on human behavior in general [141], known as the *negativity bias*.

Of course, many heuristics and tweaks (e.g., user and item biases discussed in Sec. 2.2.2 and Sec. 2.3.2) could be proposed for traditional techniques to fix the problem; however, there are intrinsic limitations within standard models that make the task hardly solvable. For example, algorithms could start looking for less similar items in the presence of an item with negative feedback.

However, there is a problem of preserving relevance to the user's tastes. It is not enough to just pick the most dissimilar items, as they are most likely to lose the connection to user preferences. Moreover, it is not even clear when to switch between the "least similar" and the "most similar" modes. If a user assigns a 3-star rating for a movie, does it mean that the system still has to look for the least similar items or should it switch back to the most similar ones? User-based similarity approach is also problematic, as it tends to generate a very broad set of recommendations with a mix of similar and dissimilar items, which again leads to the problem of extracting the most relevant, yet unlike recommendations.

In order to deal with the denoted problems, we propose a new tensor-based model that treats feedback data as a special type of categorical variable. We show that our approach not only helps to improve user cold start scenarios but also increases general recommendations accuracy.

5.1 Problem formulation

The goal of conventional recommender system is to be able to accurately generate a personalized list of new and interesting items (*top-n recommendations*), given a sufficient number of examples with known user preferences. As has been noted, if preferences are unknown this requires special techniques, such as rating elicitation, to be involved first, which in turn may disappoint users. In order to avoid that extra step or at least make it less frustrating we introduce the following additional requirements for a recommender system:

- the system must be sensitive to a full user feedback scale and do not disregard its negative part,
- the system must be able to respond properly even to a single feedback and take into account its type (positive or negative).

These requirements should help to gently navigate new users through the catalog of items, making an overall experience more personalized, as with every new feedback the system will gradually improve preference predictions.

5.1.1 Limitations of standard models

Let us consider without the loss of generality the problem of movie recommendations. Traditionally, this is formulated as a rating prediction task:

$$f_u : \text{User} \times \text{Movie} \rightarrow \text{Rating}, \tag{5.1}$$

where User is a set of all users, Movie is a set of all movies and f_u is a utility function, which assigns predicted values of ratings to every (*user, movie*) pair. The utility function in CF models is learned from a prior history of interactions, i.e. previous examples of how users rate movies, which can be conveniently represented in the form of a matrix $A \in \mathbb{R}^{M \times N}$ with M rows corresponding to the number of users and N columns corresponding to the number of movies. Elements a_{ij} of the matrix A denote actual movie ratings assigned by users. As users tend to provide feedback only for a small set of movies, not all entries of A are known, and the utility function is expected to infer the missing values.

In order to provide recommendations, the *inferred values of ratings* $R = [r_{ij}]_{i,j=1}^{M,N}$ are used to rank movies and build a ranked list of top-n recommendations, which in the simplest case is generated as:

$$\text{toprec}(i, n) := \arg\max_{j \in \text{Movie}}^{n} r_{ij}. \tag{5.2}$$

where $\text{toprec}(i, n)$ is a list of n top-ranked movies predicted for user i. The way values r_{ij} are calculated depends on a CF algorithm and we argue that *standard algorithms are unable to accurately predict relevant movies given a user with only low ratings in his or her preferences.*

Example with matrix factorization

Let us first start with the MF approach. As we do not aim at predicting the exact values of ratings and are more interested in correct ranking, it is adequate to employ

the PureSVD model of some rank $r < \min(M, N)$ described in Sec. 2.2.1. The matrix of predictions in that case can be expressed as:

$$R = U\Sigma V^T \equiv PQ^T, \tag{5.3}$$

where we also provide an equivalent form with factors $P = U\Sigma^{\frac{1}{2}}$ and $Q = V\Sigma^{\frac{1}{2}}$, commonly used in other matrix factorization techniques.

In contrast to many other MF techniques, one of the key properties of SVD is an orthogonality of columns in factor matrices U and V. As it was shown in Sec. 2.2.3, this property helps to find approximate values of ratings even for users that were not a part of the original matrix A with a very simple and easy to compute expression:

$$\boldsymbol{r} \approx VV^T\boldsymbol{a}, \tag{5.4}$$

where $\boldsymbol{a} \in \mathbb{R}^N$ is a *sparse* vector of known user preferences, where a position of every non-zero element corresponds to a movie, rated by a new user, and its value corresponds to the actual user rating on that movie. Respectively, $\boldsymbol{r} \in \mathbb{R}^N$ is a dense vector of all predicted movie ratings.

Nevertheless, there is a subtle issue here. If, for instance, $\boldsymbol{a}$ contains only a single rating, then it does not matter what exact value it has. Different values of the rating will simply scale all the resulting scores, given by Eq. (5.4), and will not affect the actual ranking of recommendations. In other words, *the recommendation list in the case of a single 2-star rating is going to be the same as in the case a 5-star rating.* In the case of a well-known user with a lot of known ratings this effect can be less pronounced. However, when preference information is limited, like in the case with new users in the cold start setting, this effect may play a crucial role in the willingness of a user to continue using such an insensitive recommendation service.

Example with similarity-based approach

It may seem that the problem can be alleviated if we exploit some user similarity technique with the help of a user-based *kNN* approach mentioned in Sec. 1.1.2. Indeed, if users share not only what they like, but also what they dislike, then users, similar to the one with a negative only feedback, might give a good list of candidate movies for

Table 5.1: Similarity-based recommendations issue.

	Scarface	Toy Story	Godfather
		Observation	
Alice	2	5	3
Bob	4		5
Carol	2	5	
		New user	
Tom	2	?	?
		Prediction	
		2.6	**3.1**

recommendations. The list can be generated with help of the following expression:

$$r_{ij} = \frac{1}{K} \sum_{k \in \mathcal{N}_i} r_{kj} \ \mathrm{sim}(i,k), \tag{5.5}$$

where $\mathcal{N}_i$ is a set of users, the most similar to user i, $\mathrm{sim}(i,k)$ is some similarity measure between users i and k; K is a normalizing factor, equal to $\sum_{k \in \mathcal{N}_i} |\mathrm{sim}(i,k)|$ in the simplest case. Similarity between users can be computed by comparing either their latent features given by the rows of matrix U or simply by the preferences encoded in the rows of the initial matrix A. It can be also modified to a more advanced forms, which take into account user biases [2].

However, even though more relevant items are likely to get higher scores with this user-similarity approach, it still does not guarantee an isolation of irrelevant items. Let us demonstrate it on a simple example. For illustration purposes we will use a simple kNN approach, based on a cosine similarity measure. However, it can be generalized to more advanced variations with different similarity measures as well.

Let a new user Tom have rated the “Scarface” movie with rating 2 (see Table 5.1) and we need to decide which of two other movies, namely “Toy Story” or “Godfather”, should be recommended to Tom, given an information on how other users – Alice, Bob and Carol – have also rated these movies. As it can be seen, Alice and Carol, similarly to Tom, do not like criminal movies. They also both enjoy the “Toy Story” animation. Even though Bob demonstrates an opposite set of interests, the preferences of Alice and Carol prevail. From here it can be concluded that the most relevant (or

safe) recommendation for Tom would be the "Toy Story". Nevertheless, the prediction formula Eq. (5.5) assigns the highest score to the "Godfather" movie, which is the result of a higher value of cosine similarity between Bob's and Tom's preference vectors.

5.1.2 Resolving the inconsistencies

The problems described above suggest that in order to build a model, which fulfills the requirements, proposed in the beginning of Sec. 5.1, we have to move away from traditional representation of ratings. Our idea is to restate the problem formulation in the following way:

$$f_u : \text{User} \times \text{Movie} \times \text{Rating} \rightarrow \text{Relevance Score}, \tag{5.6}$$

where Rating is a domain of ordinal (categorical) variables, consisting of all possible user ratings, and Relevance Score denotes the likeliness of observing a certain (*user, movie, rating*) triplet. With this formulation relations between users, movies and ratings are modelled in a ternary way, i.e. all three variables influence each other and the resulting score.

This type of relations can be modelled with several methods, such as Factorization Machines [136] or other context-aware methods [3]. We propose to solve the problem with a tensor-based approach, as it seems to be more flexible and naturally fits the formulation in Eq. (5.6) (see Fig. 5.1). More formally, with this approach the observed (*user, movie, rating*) triplets are encoded within a third order tensor $\mathcal{A} \in \mathbb{R}^{M \times N \times K}$ with M, N and K corresponding to the total number of unique users, movies and ratings respectively. We set the values of $\mathcal{A}$ to be binary:

$$\begin{cases} a_{ijk} = 1, & \text{if } (i,j,k) \in S, \\ a_{ijk} = 0, & \text{otherwise}, \end{cases} \tag{5.7}$$

where S is a history of known interactions, i.e. a set of the observed (*user, movie, rating*) triplets. Similarly to the MF case represented by Eq. (5.3), we are interested in finding such a tensor factorization that helps to reveal common patterns in data and to build a latent representation of users, movies and additionally ratings.

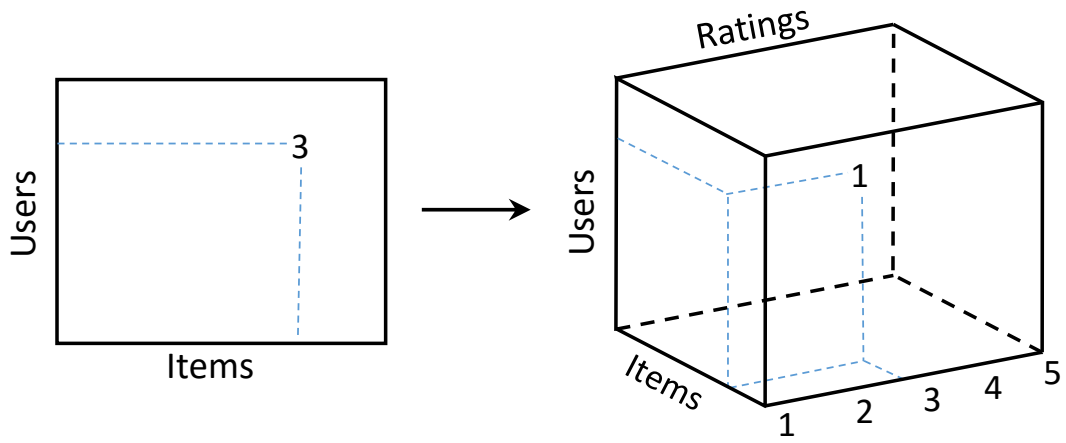

Figure 5.1: From a matrix to a third order tensor.

5.2 Proposed approach

The corresponding tensor approximation problem can be formulated in terms of minimization of the following square loss function:

$$\mathscr{L}(\mathcal{A}, \mathcal{R}) = \|\mathcal{A} - \mathcal{R}\|_{\mathrm{F}}^2, \tag{5.8}$$

where $\mathcal{R}$ is a low rank tensor approximation and $\|\cdot\|_{\mathrm{F}}$ denotes Frobenius norm similarly to the matrix case, i.e. $\|\mathcal{X}\|_{\mathrm{F}}^2 = \sum_{i_1} \sum_{i_2} \dots \sum_{i_d} x_{i_1 i_2 \dots i_d}^2$.

Both CP and TD decompositions are suitable for this task. However, following the SVD-based approach we additionally require orthogonality of columns in factor matrices. Later we will show that this requirement allows for quick computation of recommendations, similarly to Eq. (5.4) in the SVD case.

The orthogonality constraint can be naturally satisfied with the TD decomposition, hence we seek for the solution in the following format (see Sec. 3.1.3):

$$\mathcal{R} = \mathcal{G} \times_1 U \times_2 V \times_3 W,$$

where $\mathcal{G} \in \mathbb{R}^{r_1 \times r_2 \times r_3}$ is the core of decomposition; $U \in \mathbb{R}^{M \times r_1}, V \in \mathbb{R}^{N \times r_2}, W \in \mathbb{R}^{K \times r_3}$ are *columnwise orthogonal* factor matrices. The first two of them represent embedding of users and items onto a reduced latent feature space, similarly to the SVD case. The third matrix W gives an additional latent representation of ratings. The decomposition is computed with the help of HOOI algorithm described in Sec. 3.1.4.

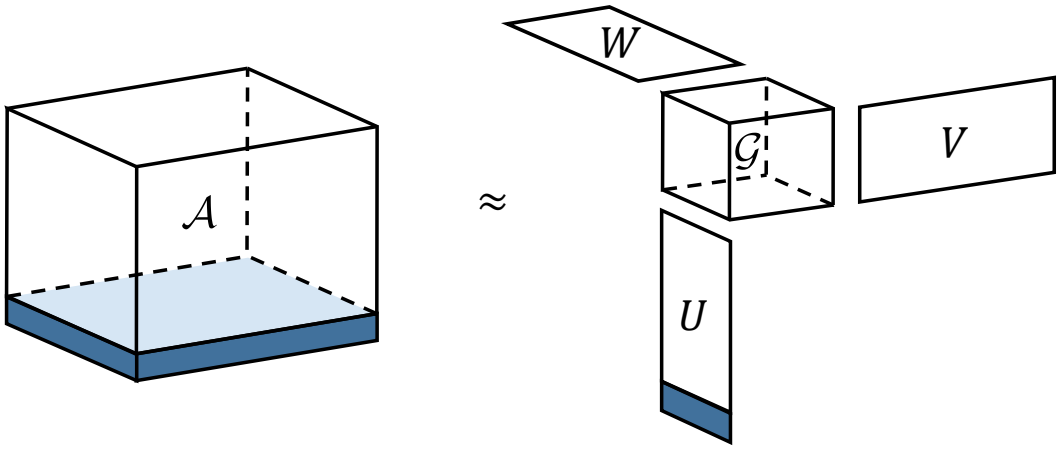

Figure 5.2: Higher order folding-in for Tucker decomposition. A slice with a new user information in the original data (left) and a corresponding row update of the factor matrix in Tucker decomposition (right) are marked with solid a color.

5.2.1 Efficient computation of recommendations

Recommender systems typically have to deal with large number of users and items, which renders the problem of fast computation of recommendations. For example, factorizing the tensor for every new user can take prohibitively long time, which is inconsistent with the requirement of real-time engagement with users. In order to address that problem we propose a *higher order folding-in* method (see Fig. 5.2). Similarly to the SVD case, it helps to find approximate recommendations for any unseen user with comparatively low computational cost (cf. Eq. (5.4)):

$$R_{(i)} = V V^T P_{(i)} W W^T, \tag{5.9}$$

where $P_{(i)}$ is an $N \times K$ binary matrix that encodes preferences of an user i and $R_{(i)} \in \mathbb{R}^{N \times K}$ is a relevance score prediction matrix. Similarly to the SVD-based folding-in given by Eq. (5.9), it can be treated as a sequence of orthogonal projections to latent spaces of movies and ratings.

5.2.2 Shades of ratings

Note that even though Eq. (5.9) looks very similar to Eq. (5.4), there is a substantial difference in what is being scored. In the case of a matrix factorization we score ratings (or other forms of feedback) directly, whereas in the tensor case we *score the likeliness of a rating to have a particular value for an item.* This gives a new and more

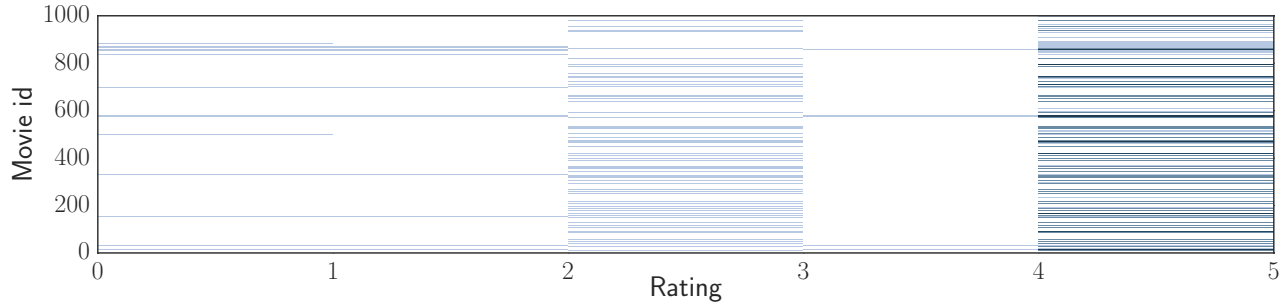

Figure 5.3: An example of predicted user preferences that we call *shades of ratings*. Every horizontal bar can be treated as a likeliness of some movie to have a specific rating for a selected user. More dense colors correspond to higher relevance scores.

informative view on the predicted user preferences (see Fig. 5.3). Unlike the conventional methods, every movie in recommendations is associated not just with a single score, but rather with a full range of all possible rating values that users are exposed to.

Another remarkable property of "rating shades" is that *it can be naturally utilized for both ranking and rating prediction tasks*. The ranking task corresponds to finding a maximum score along the movies mode (2nd mode of the tensor) for a selected (highest) rating. For example, in the case of 5-star scale, the task can be expressed as:

$$\text{toprec}(i, n) := \underset{j \in \text{Movie};\, k=5}{\arg\overset{n}{\max}}\ r_{ijk}. \tag{5.10}$$

We also note that ranking can be performed within every rating value. Moreover, if a positive feedback is defined by several ratings (e.g. 5 and 4), then the sum of scores from these ratings can be used for ranking as well:

$$\text{toprec}(i, n) := \underset{j \in \text{Movie}}{\arg\overset{n}{\max}} \sum_{k \geq k_p} r_{ijk},$$

where k_p is a positivity threshold, e.g. $k_p = 4$. Our experiments show that this typically leads to an improved quality of predictions comparing to Eq. (5.10).

Rating prediction can be performed in a similar fashion as it simply corresponds to maximization of relevance scores along the ratings mode (i.e. the 3rd mode of the tensor) for a selected movie.

5.3 Evaluation

As has been discussed in Sec. 5.1.1, standard recommender models are unable to properly operate with a negative feedback and more often simply ignore it. As an example, a well known recommender systems library *MyMediaLite* [59] that features many state-of-the-art algorithms, does not support a negative feedback for item recommendation tasks.

In addition to that, a common way of performing an offline evaluation of recommendations' quality is to measure only how well a tested algorithm can retrieve highly relevant items. Nevertheless, both relevance-based (e.g. precision, recall) and ranking-based (e.g. nDCG, MAP) metrics, are completely insensitive to irrelevant items prediction: an algorithm that recommends 3 positively rated and 7 negatively rated items will gain the same evaluation score as an algorithm that recommends 3 positively rated and 7 items with unknown (not necessarily negative) ratings.

This leads to several important questions, which are typically obscured and which we aim to find an answer to:

- How likely is an algorithm to place irrelevant items in top-n recommendation list and rank them highly?
- Does high evaluation performance in terms of relevant items prediction guarantee a lower number of irrelevant recommendations?

Answering these questions is impossible within standard evaluation paradigm and we propose to adopt commonly used metrics in a way that respects crucial difference between the effects of relevant and irrelevant recommendations. We also expect that modified metrics will reflect the effects, described in the beginning of this chapter (the Scarface and Godfather example).

5.3.1 Negativity bias compliance

The first step for a proper modification of metrics is to split rating values into 2 classes: the *negative feedback* class and the *positive feedback* class. This is achieved by selecting a *positivity threshold* value, such that the values of ratings equal to or above this threshold are treated as positive examples and all other values – as negative.

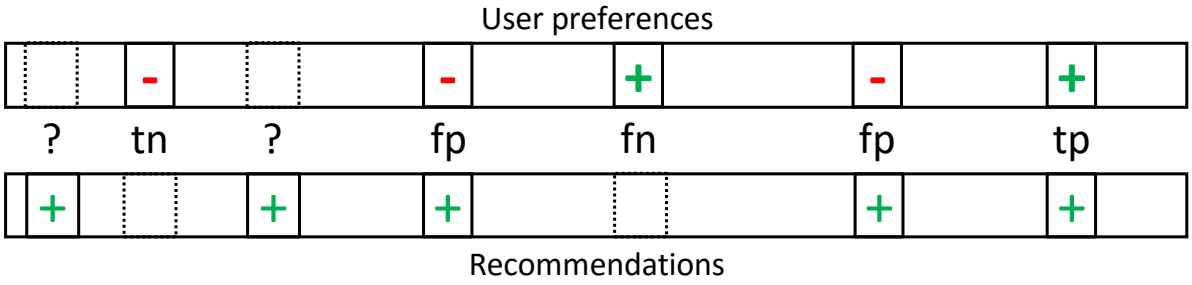

Figure 5.4: Definition of matching and mismatching predictions. Recommendations that are not a part of the known user preferences (question marks) are ignored and not considered as false positive.

The next step is to allow generated recommendations to be evaluated against the negative user feedback, as well as the positive one. This leads to a classical notion of true positive (tp), true negative (tn), false positive (fp) and false negative (fn) types of predictions [153], which also renders a classical definition of relevance metrics, namely precision (P) and recall (R, also referred as *True Positive Rate* (TPR)):

$$\mathrm{P} = \frac{\mathrm{tp}}{\mathrm{tp}+\mathrm{fp}}, \quad \mathrm{R} = \frac{\mathrm{tp}}{\mathrm{tp}+\mathrm{fn}}.$$

Similarly, *False Positive Rate* (FPR) is defined as

$$\mathrm{FPR} = \frac{\mathrm{fp}}{\mathrm{tp}+\mathrm{fp}}.$$

The TPR to FPR curve, also known as a Receiver Operating Characteristics (ROC) curve, can be used to assess the tendency of an algorithm to recommend irrelevant items. Worth noting here that if items, recommended by an algorithm, are not rated by a user (question marks on Fig. 5.4), then we simply ignore them and do not mark as false positive in order to avoid fp rate overestimation [153].

The *Discounted Cumulative Gain* (DCG) metric will look very similar to the original one with the exception that we do not include the negative ratings into the calculations at all:

$$\mathrm{DCG} = \sum_{p} \frac{2^{r_p}-1}{\log_2(p+1)}, \tag{5.11}$$

where $p : \{r_p \geq \text{positivity threshold}\}$ and r_p is a rating of a positively rated item. This gives an nDCG metric:

$$\mathrm{nDCG} = \frac{\mathrm{DCG}}{\mathrm{iDCG}},$$

where iDCG is a value returned by an ideal ordering or recommended items (i.e. when more relevant items are ranked higher in top-n recommendations list).

5.3.2 Penalizing irrelevant recommendations

The nDCG metric indicates how close tp predictions are to the beginning of a top-n recommendation list, however, it tells nothing about the ranking of irrelevant items. We fix this by a modification of Eq. (5.11) with respect to a negative feedback, which we call a *Discounted Cumulative Loss*:

$$\mathrm{DCL} = \sum_{n} \frac{2^{-r_n} - 1}{-\log_2(n+1)}, \tag{5.12}$$

where $n : \{r_n < \text{positivity threshold}\}$ and r_n is a rating of a negatively rated item. Similarly to nDCG, nDCL metric is defined as:

$$\mathrm{nDCL} = \frac{\mathrm{DCL}}{\mathrm{iDCL}},$$

where iDCL is a value returned by an ideal ranking or irrelevant predictions (i.e. when more irrelevant items are ranked lower). Note that as nDCL penalizes high ranking of irrelevant items, *the lower are the values of nDCL the better.*

In the experiments all the metrics are measured for different values of top-n list length, i.e. the metrics are metrics *at n* denoted as @n. The values of metrics are averaged over all test users.

5.3.3 Evaluation methodology

For the evaluation purposes we split datasets into two subsets, disjoint by users (e.g. every user can only belong to a single subset). First subset is used for learning a model, it contains 80% of all users and is called a *training set.* The remaining 20% of users (the test users) are unseen in the training set and are used for models' evaluation. We holdout a fixed number of items from every test user and put them into a *holdout set.* The remaining items form an *observation set* of the test users. Recommendations, generated based on an observation set are evaluated against the holdout set.

The main difference from common evaluation methodologies is that *we allow both relevant and irrelevant items in the holdout set* (see Fig. 5.4). Furthermore, we

vary the number and the type of items in the observation set, which leads to various scenarios:

- leaving only one or few items with the lowest ratings leads to the case of "no-positive-feedback" cold start;
- if all the observation set items are used to predict user preferences, this serves as a proxy to a standard recommendation scenario for a known user.

Using this scheme we perform a 5-fold cross validation by selecting different 20% of users each time and averaging the results across all 5 folds.

5.4 Experimental setup

This section describes various settings, including dataset preprocessing, selection of algorithms and problem specific modifications to evaluation methodology.

5.4.1 Datasets

We use publicly available Movielens[1] 1M and 10M datasets as a common standard for offline recommender systems evaluation. We have also trained a few models on the latest Movielens dataset (22M rating, updated on 1/2016) and deployed a movie recommendations web application for online evaluation. This is especially handy for our specific scenarios, as the content of each movie is easily understood and contradictions in recommendations can be easily eye spotted (see Table 5.2). We preprocess these datasets to contain only users who have rated no less than 20 movies.

5.4.2 Algorithms

We compare our approach with the state-of-the-art matrix factorization methods, designed for item recommendations task as well as two non-personalized baselines.

- *CoFFee* (Collaborative Full Feedback model) is the proposed tensor-based approach.

[1] https://grouplens.org/datasets/movielens/

- *SVD*, also referred as *PureSVD* described in Sec. 2.2.1. Missing values in this model are simply imputed with zeros.
- *WRMF* is the MF method, described in Sec. 2.3.3, where missing values are uniformly weighted.
- *BPRMF* [135] is a matrix factorization method, powered by a Bayesian personalized ranking approach (BPR), which optimizes pairwise preferences between observed and unobserved items.
- *Most popular* model always recommends top-n items with the highest number of ratings (independently of ratings value).
- *Random guess* model generates recommendations randomly.

SVD is based on Python's Numpy, and SciPy packages, which heavily use BLAS and LAPACK functions as well as MKL optimizations. CoFFee is also implemented in Python, using the same packages for most of the linear algebra operations. We additionally use Pandas package to support sparse tensors in COO format.

BPRMF and WRMF implementations are taken from the MyMediaLite [59] library. Only positively rated data is used for training these two models. In the case of BPRMF it is additionally binarized. We wrap the command line utilities of these methods with Python, so that all the tested algorithms share the same namespace and configuration. Command line utilities do not support online evaluation and we implement our own (orthogonalized) folding-in on the factor matrices generated by these models. Learning times of the models are depicted on Fig. 5.5. The source code as well as the link to our web app can be found at Github[2].

5.4.3 Settings

The number of holdout items is always set to 10. The top-n values range from 1 to 100. We perform 3 different selection mechanisms for observation set. In the first one we sample either 1 or 3 *negatively rated* items from test user preferences. In the second scheme we select 1, 3 or 5 items *at random*. Finally, in the third case we select *all available* test user items (see Sec. 5.3.3 for details).

[2]https://github.com/Evfro/fifty-shades

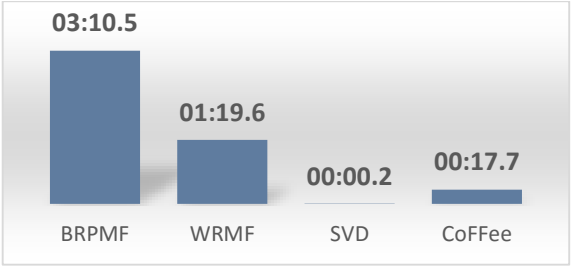

Figure 5.5: Models' learning time, mm:ss.ms (single laptop, Intel i5 2.5GHz CPU, Movielens 10M).

We include higher values of top-n (up to 100) as we allow random items to appear in the holdout set. This helps to make experimentation more sensitive to wrong recommendations that match negative feedback from users. Both observation and holdout sets are cleaned from the items that are not present in the training set. The number of latent factors for all matrix factorization models is set to 10, CoFFee multilinear rank is (13, 10, 2). Regularization parameters of WRMF and BPRMF algorithms are set to MyMediaLite's defaults.

The positivity threshold is set to 4 for both Movielens 1M and Movielens 10M. Worth noting here that lower values of positivity threshold (e.g., 3 or 2) mostly lead to the changes in absolute numbers, while the general behavior of models stays approximately the same with only minor rearrangements. Due to this reason we only report results for one fixed value of the positivity threshold.

5.5 Results

Evaluation results are presented on Fig. 5.6. Rows A and C correspond to Movielens 1M dataset, rows B and D correspond to Movielens 10M dataset. We also report a few interesting hand-picked examples of movies recommendations, generated from a single negative feedback (see Table 5.2).

How to read the graphs. The results are better understood with particular examples. Let us start with the first two rows on Fig. 5.6 (row A is for Movielens 1M and row B is for Movielens 10M). These rows correspond to a performance of the models, when only a single (random) negative feedback is provided.

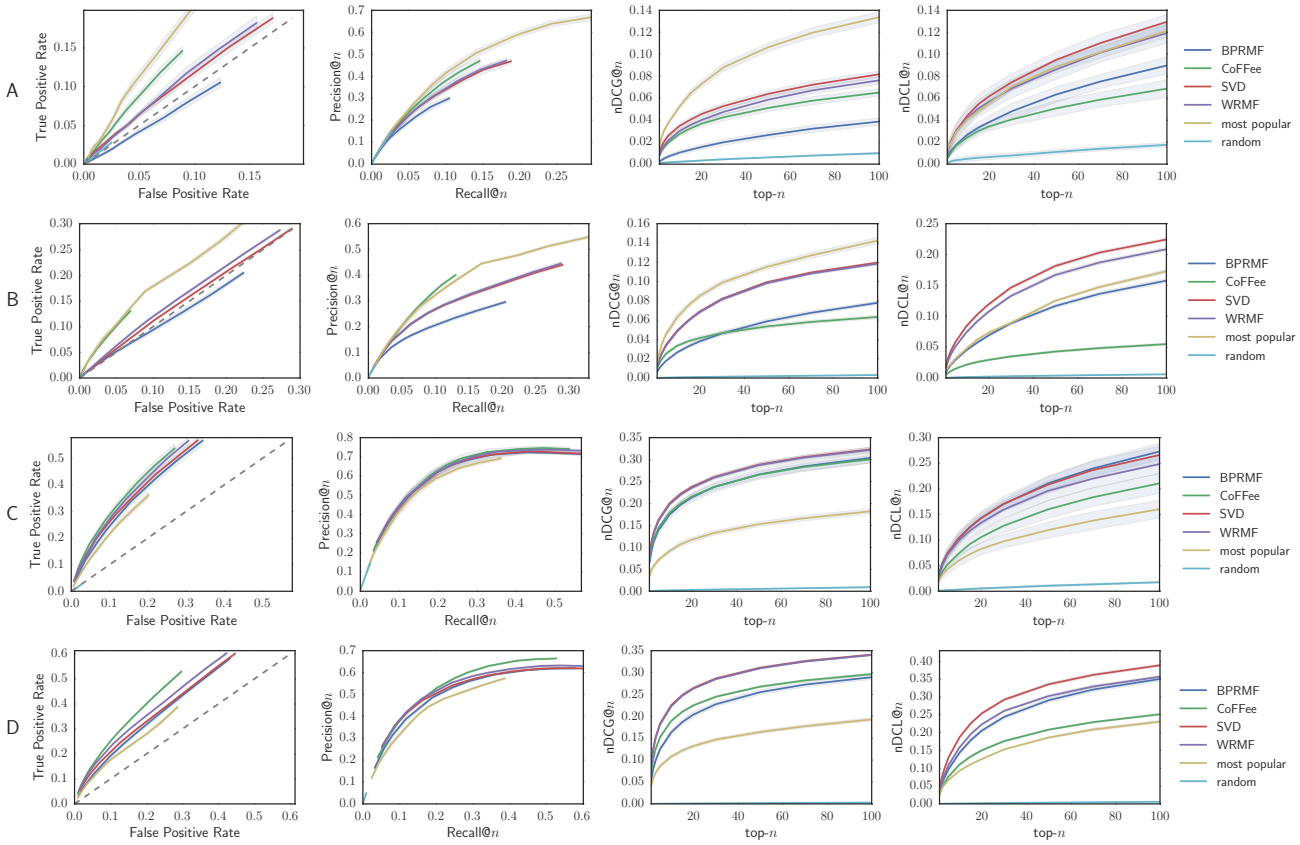

Figure 5.6: The ROC curves (1st column), precision-recall curves (2nd column), nDCG@n (3rd column) and nDCL@n (4th column). Rows A, B correspond to a cold start with a *single negative* feedback. Rows C, D correspond to a *known user* recommendation scenario. Odd rows are for Movielens 1M, even rows are for Movielens 10M. For the first 3 columns the higher the curve, the better, for the last column the lower the curve, the better. Shaded areas show a standard deviation of an averaged over all cross validation runs value.

First of all, it can be seen that the item popularity model gets very high scores with TPR to FPR, precision-recall and nDCG metrics (first 3 columns on the figure). One could conclude that this is the most appropriate model in that case (when almost nothing is know about user preferences). However, high nDCL score (4th column) of this model indicates that it is also very likely to place irrelevant items at the first place, which can be disappointing for users. Similar poor ranking of irrelevant items is observed with SVD and WRMF models. On the other hand, the lowest nDCL score belongs to the random guess model, which is trivially due to a very poor overall per-

Table 5.2: Hand-picked examples from top-10 recommendations generated on a single feedback. The models are trained on the latest Movielens dataset.

	Scarface ★★☆☆☆	LOTR: The Two Towers ★★☆☆☆	Star Wars: Episode VII - The Force Awakens ★★★★★
CoFFee	Toy Story	Net, The	Dark Knight, The
	Mr. Holland's Opus	Cliffhanger	Batman Begins
	Independence Day	Batman Forever	Star Wars: Episode IV - A New Hope
SVD	Reservoir Dogs	LOTR: The Fellowship of the Ring	Dark Knight, The
	Goodfellas	Shrek	Inception
	Godfather: Part II, The	LOTR: The Return of the King	Iron Man

formance. The same conclusion is valid for BPRMF model, which has low nDCL (row A), but fails to recommend relevant items from a negative feedback.

The only model that behaves reasonably well is the proposed CoFFee model. It has low nDCL, i.e. *it is more successful at avoiding irrelevant recommendations at the first place.* This effect is especially strong on the Movielens 10M dataset (row B). The model also exhibits a better or comparable to the item popularity model's performance on relevant items prediction. At first glance, the surprising fact is that the model has a low nDCG score. Considering the fact that it can not be due to a higher ranking of irrelevant items (as it follows from low nDCL), this is simply due to a higher ranking of items, which were not yet rated by a user (recall the question marks on Fig. 5.4).

The model makes a *safe guess* by filtering out irrelevant items and proposing those items that are more likely to be relevant to an original negative feedback (unlike popular or similar items recommendation). This conclusion is also supported by the examples from the first 2 columns of Table 5.2. It can be easily seen that, unlike SVD, the CoFFee model makes safe recommendations with "opposite" movie features (e.g. Toy Story against Scarface). Such an effects are not captured by standard metrics and can be revealed only by a side by side comparison with the proposed nDCL measure.

In standard recommendations scenario, when user preferences are known (rows C, D on Fig. 5.6) our model also demonstrates the best performance in all but nDCG metrics, which again is explained by the presence of unrated items rather than a poor quality of recommendations. In contrast, MF models – SVD and WRMF – while also being the top-performers in the first three metrics, demonstrate the worst quality in terms of nDCL almost in all cases.

5.6 Conclusion and perspectives

To conclude, let us first address the two questions, posed at the beginning of Sec. 5.3. As we have shown, standard evaluation metrics that do not treat irrelevant recommendations properly (as in the case with nDCG), might obscure a significant part of a model's performance. An algorithm that highly scores both relevant and irrelevant items is more likely to be favored by such metrics while increasing the risk of a user disappointment.

We have proposed modifications to both standard metrics and evaluation procedure, which not only reveal a positivity bias of standard evaluation but also help to perform a comprehensive examination of recommendations' quality from the perspective of both positive and negative effects.

We have also proposed a new model that is able to learn equally well from full spectrum of user feedbacks and provides state-of-the-art quality in different recommendation scenarios. The model is unified in a sense that it can be used both at an initial step of learning user preferences and at standard recommendation scenarios for already known users. We believe that the model can be used to complement or even replace standard rating elicitation procedures and help to safely introduce new users to a recommender system, providing highly personalized experience from the very beginning.

Chapter 6

Hybrid factorization model

Since the very beginning of the recommender systems field, there was active research devoted to various dimensionality reduction methods allowing to build more expressive and more accurate latent factor models. A considerable part of this research was specifically devoted to elaborate MF techniques. As we have seen in Chap. 2, these techniques offer a very flexible framework for addressing many different aspects of the recommendation task.

The interest in the MF approach had been additionally warmed up by the famous Netflix prize competition. However, one of its main critiques was a narrow focus on rating prediction. As has been noted in Sec. 1.2.4, good rating prediction performance cannot be straightforwardly translated to a good performance in terms of the top-n recommendations task. In fact, one of the simplest SVD-based models called *PureSVD*, which is not even suitable for rating prediction, was proven to outperform other much more sophisticated MF algorithms (see Sec. 2.2.1).

Moreover, as we have discussed in Sec. 2.5 and summarized in Sec. 4.4, PureSVD offers a number of practical advantages, such as global convergence with deterministic output backed by solid linear algebra, highly optimized implementations based on BLAS and LAPACK routines, a lightweight hyper-parameter tuning achieved by a simple rank truncation, analytical expression for instant online recommendations, scalable modifications based on randomized algorithms. Therefore, we find it necessary to distinguish this approach from all other MF methods. Even though some of them are inspired by SVD and have its acronym in their names (e.g., FunkSVD, SVD++, etc.), they are not SVD-based and do not provide the same set of advantages.

However, as any other CF technique, PureSVD relies solely on the knowledge about user preferences expressed in the form of ratings, likes, purchases or other types of feedback, either explicit or implicit. On the other hand, as has been discussed in Chap. 4, this information may not always be sufficient for learning a representative model. If interaction data is too scarce, it may become extremely difficult to discover reliable patterns from the observations without considering additional sources of knowledge about users and items.

Addressing the described problems of insufficient preference data has lead to the development of hybrid models, introduced in Sec. 1.1.3. As we have also discussed in Sec. 4.3, a significant body of work is specifically devoted to incorporating side information into MF methods. Surprisingly, the SVD-based approach has received much lower attention in this regard, despite having many practical advantages.

It was shown to be a convenient tool for factorizing aggregated representations of feature matrices and collaborative data [166]. However, in this case, the structure of interactions vanishes due to the aggregation process and the obtained factors can only be used as an *intermediate result*. Therefore, unlike PureSVD, it requires some other CF algorithm for generating predictions and leads to a more complicated solution.

To the best of our knowledge, there were no attempts for developing an integrated hybrid approach, where *interaction data and side information would be jointly factorized with the help of SVD*, and the obtained result would be used as an *end model* that allows to immediately generate recommendations as in the PureSVD approach. With this work, we aim to fill that gap and extend the family of hybrid methods with a new approach based on a straightforward modification of PureSVD.

6.1 Understanding the limitations of SVD

As we have demonstrated in Chap. 2, MF methods offer a great level of flexibility allowing to tackle various RS problems and fine-tune a desired solution. This includes the already mentioned ability to blend both interaction data and additional side knowledge within a single optimization objective, which, among other benefits, produces a more meaningful latent feature space with a certain inner structure, controlled by side information.

Table 6.1: An example of insufficient preference data problem

	Item1	Item2	Item3	Item4	Item5
			Observed interactions		
Alice	1		1	1	
Bob	1	1		1	
Carol	1			1	
			New user		
Tom	1	?	?	1	1
PureSVD:		0.3	0.3		
Our approach:		0.1	0.6		

Table note: Item5 is a cold-start item. *Item3* and *Item5*, highlighted with blue color, are assumed to be similar to each other in terms of their characteristics. This assumption is reflected in the *Our approach* row. The *PureSVD* row corresponds to the PureSVD model of rank 2. The code to reproduce this result can be found at https://gist.github.com/Evfro/c6ec2b954adfff6aaa356f9b3124b1d2.

This technique, however, is not available off-the-shelf in the PureSVD approach due to the classical formulation of the truncated SVD problem with its fixed optimization objective given by Eq. (2.4). In this work we aim to find a new way to formulate the optimization problem so that, *while staying within the same computational paradigm of the truncated SVD*, it would allow us *to account for additional sources of information* during the optimization process. In order to do this we first need to decompose SVD internals and see what exactly affects the formation of its latent feature space.

6.1.1 When PureSVD does not work

Consider the following simple example on fictitious interaction data depicted in Table 6.1. Initially we have 3 users (Alice, Bob and Carol) and 5 items, with only first 4 items being observed in interactions (the first 3 rows and 4 columns of the table). The last column corresponds to a cold start (i.e. previously unobserved) *Item5*. We use this toy data to build PureSVD of rank 2 and generate recommendations for a new user Tom (*New user* row in the table), who has already interacted with *Item1*, *Item4* and *Item5*.

Let us suppose that in addition to interaction data we are also provided with some prior knowledge about item relations. More specifically, we assume that *Item3 and Item5*, highlighted with blue color, *are more similar in their characteristics to each other* than to other items. For example, they can be of the same unique category. In that case, since Tom has expressed an interest in *Item5*, it seems natural to expect from a good recommender system to favor *Item3* over *Item2* in recommendations. This, however, does not happen with PureSVD.

With the help of the *folding-in* technique given by Eq. (2.11) it can be easily verified that the scores predicted by SVD will be equal for both items as shown next to the *PureSVD* entry in the bottom of Table 6.1. This example demonstrates a general limitation of the PureSVD approach related to the lack of preference data: *the lower is the density of data the harder it is for SVD to recover meaningful relations* (see Fig. 6.1). Obviously, this problem cannot be resolved without taking into account additional problem-specific knowledge. In this chapter *we show how to remove this limitation by employing side data* and help an SVD-based model generate more reasonable predictions (see *Our approach* entry in Table 6.1 as an example).

It should be also noted that if Carol would have additionally rated both *Item3* and *Item5*, this would build a connection between these items in the model and lead to the appropriate scores even without side information. This leads to an idea that depending on the sparsity of interactions, using *side information may not always provide additional benefits*. We investigate this idea in Sec. 6.4. This also opens up a perspective of addressing (at least partially) an important question, "*why SVD works well for some recommender applications, and less well for others*", raised in [146].

6.1.2 Why PureSVD does not work

The formal explanation of the observed result requires understanding of how exactly computations are performed in the SVD algorithm. A very rigorous mathematical analysis of that is performed by the authors of the EIGENREC model, described in Sec. 2.2.4. As the authors demonstrate, the *latent factor model of PureSVD can be viewed as an eigendecomposition of a scaled user-based or item-based cosine similarity matrix*.

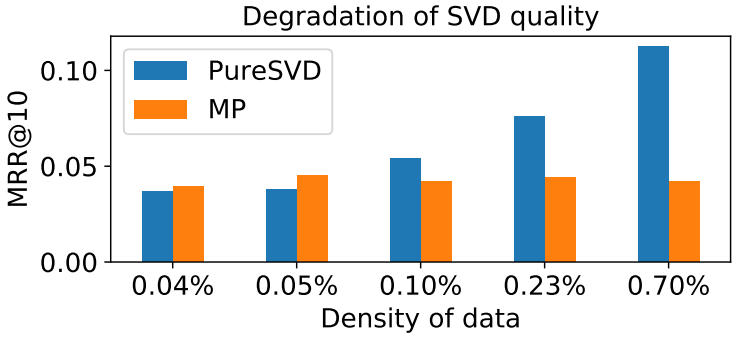

Figure 6.1: The quality of PureSVD recommendations is very sensitive to the lack of preference data and may even fall below the non-personalized popularity-based model (denoted as MP) with extreme sparsity. Results from Movielens-10M dataset (see Sec. 6.3 for details). At every sparsity level the rank of PureSVD is tuned to provide the best result.

Recall that, for example, in the user-based case each element of the similarity matrix is proportional to the scalar product between the corresponding rows of the original rating matrix:

$$c_{ij} \sim \boldsymbol{a}_i^T \boldsymbol{a}_j, \tag{6.1}$$

where $\boldsymbol{a}_i$ is a sparse vector that encodes preferences of user i. This observation immediately suggests that *any cross-item relations are simply ignored by SVD* as it takes only item co-occurrence into account. In other words, *the contribution of a particular item into the final user similarity score c_{ij} is counted only if the item is present in preferences of both user i and user j*. Similar conclusion also holds for the item-based case. This fully explains the uniform scores assigned by PureSVD in our fictitious example.

The authors of EIGENREC propose to replace the scaling factors as well as the similarity matrix with some new matrices, which fit more adequately into an underlying relations model. Among various proposed replacement options, one particular similarity measure, namely Jaccard Index, could partially solve the described problem, as to some extent it allows to account for cross-entity relations. However, it does not take into consideration how similar entities are. Indeed, depending on a set of features particular items or users can be more similar to each other than to others and therefore should have a higher contribution into a final similarity score.

6.2 Proposed approach

In order to account for cross-entity relations in a more appropriate way we have to find a different similarity measure that would consider all possible pairs of entities and allow us to fuse side information in there. A straightforward way to achieve this is to modify the inner product in Eq. (6.1) as follows:

$$c_{ij} \sim \boldsymbol{a}_i^T S \, \boldsymbol{a}_j. \tag{6.2}$$

where symmetric matrix $S \in \mathbb{R}^{N \times N}$ reflects auxiliary relations between items based on side information. This matrix be constructed in many ways. As an example, one can represent items as vectors of their real features (e.g., movie genres) and then compute cosine similarity between different pairs of these vectors in order to fill-in off-diagonal entries of S. More details about the properties of such matrices are given in Sec. 6.2.2.

Effectively, this matrix creates "virtual" links between users even if they have no common items in their preferences, i.e., have never rated the same item. Occasional links will be filtered out by dimensionality reduction, whereas more frequent ones will contribute into the model and help to reveal valuable consumption patterns. In a similar fashion we can introduce a matrix $K \in \mathbb{R}^{M \times M}$ to incorporate user-related information. We will use the term *side similarity* to denote these matrices. Their entries encode similarities between users or items based on side information, such as user attributes or item features.

6.2.1 HybridSVD

Equation (6.2) generates the following matrix cross-product:

$$A_0 S A_0^T, \tag{6.3}$$

where, as previously, A_0 denotes a rating matrix with unknown elements replaced by zeroes. We omit scaling factors, used in the EIGENREC model, to have a clearer picture of the effects related purely to side information handling. Scaling, however, adds an additional degree of freedom in model tuning and can be brought back at any time. In a similar fashion, matrix K gives

$$A_0^T K A_0. \tag{6.4}$$

These two cross-products have an effect on either rows or columns of A_0 and are independent of each other. *Our ultimate goal is to bring them together into a joint problem formulation with a single solution based on standard SVD.*

In order to achieve that, we note that if $A = U\Sigma V^T$ is an SVD of some matrix A, then an eigendecomposition of the corresponding Gram matrices reads:

$$\begin{cases} AA^T = U\Sigma^2 U^T, \\ A^T A = V\Sigma^2 V^T. \end{cases} \tag{6.5}$$

By plugging Eq. (6.3)–(6.4) into Eq. (6.5), we arrive at the following system of equations:

$$\begin{cases} A_0 S A_0^T = U\Sigma^2 U^T, \\ A_0^T K A_0 = V\Sigma^2 V^T, \end{cases} \tag{6.6}$$

where matrices U and V represent embeddings of users and items onto a *common latent feature space.*

The system of equations in Eq. (6.6) has a close connection to the *Generalized SVD* [66] and can be solved via the standard SVD of an auxiliary matrix $\widehat{A}$ [1]:

$$\widehat{A} \equiv K^{\frac{1}{2}} A_0 S^{\frac{1}{2}} = \widehat{U}\Sigma\widehat{V}^T, \tag{6.7}$$

where matrices $\widehat{U}, \Sigma$ and $\widehat{V}$ represent singular triplets similarly to the PureSVD model. Matrix Σ here is the same as in Eq. (6.6) and the connection between the auxiliary and the original latent feature space is established by the following relations:

$$\widehat{U} = K^{\frac{1}{2}} U, \quad \widehat{V} = S^{\frac{1}{2}} V. \tag{6.8}$$

There are certain technical challenges, related to the computation of matrix square roots in Eq. (6.7) and the need to calculate their inverses for finding U and V in Eq. (6.8). In Sec. 6.2.3 we show how to avoid these problems with the help of Cholesky decomposition. Meanwhile, let us fix the fact that *solving the joint problem of Eq. (6.6) turns out to be as simple as finding standard SVD of an auxiliary matrix* $\widehat{A}$. We call this model *HybridSVD*. As it can be seen, matrix $\widehat{A}$ "absorbs" additional relations encoded by matrices K and S and allows to model them jointly.

Orthogonality of factor matrices $\widehat{U}$ and $\widehat{V}$ allows to shed the light on the structure of factors U and V. Using the fact that $\widehat{U}^T\widehat{U} = \widehat{V}^T\widehat{V} = I$, one gets:

$$U^T K U = V^T S V = I,$$

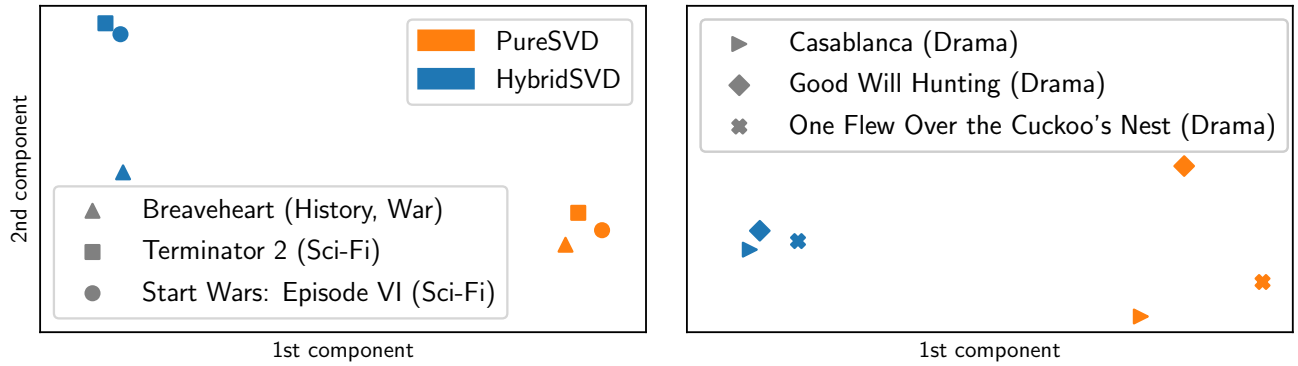

Figure 6.2: The effect of genre-based movie similarity. Scatter points correspond to different movies in the latent feature space (the first two principal components). Our model tends to pull movies of different genres apart and place movies of the same genre close to each other. Uniform, distance-preserving scaling is applied to make models comparable.

i.e. columns of the matrices U and V are orthogonal under the constraints imposed by matrices K and S respectively[1]. In other words, the structure of the latent space is directly shaped by side information. We also note that *enitities with high similarity in their own feature space are likely to become closer to each other in the latent feature space of HybridSVD* (see Figure 6.2).

6.2.2 Side similarity

Construction of the matrices K and S to a certain extent is a feature engineering task and therefore, it is difficult to provide a universal recipe. We have already mentioned one possible way based on a simple cosine similarity calculation. Indeed, there can be many other ways. However, we will limit possible options by restricting these matrices to be:

1. symmetric: $K = K^T$, $S = S^T$,
2. positive definite: $K \succ 0$, $S \succ 0$.

Cases when the above requirements do not hold are out of the scope of this work as the problem becomes more complex and computationally infeasible.

[1]This property of matrices U and V is sometimes called K- and S-orthogonality [162].

The first requirement is typically easily satisfied, however the second one is more restrictive since side information may come from heterogeneous sources and may have an arbitrary structure. In order to resolve the uncertainty we impose the following form on side similarity matrices:

$$\begin{cases} S = I + \alpha Z, \\ K = I + \beta W, \end{cases} \tag{6.9}$$

where Z, W are *zero-diagonal* real symmetric matrices with elements satisfying $-1 \leq z_{ij}, w_{ij} \leq 1 \; \forall i, j$, and α, $\beta \in \mathbb{R}^+$ are free model parameters. Note that with $\alpha = \beta = 0$ the model turns back into PureSVD.

When side similarity matrices are not strictly positive definite reducing the values of α and β allows to fix that[2]. Additional benefit provided by α and β is the ability to *control an overall contribution of side information in the model* and avoid undesirable dominance of feature-based relations over co-occurrence patterns.

In our experiments we used a simple procedure to construct similarity matrices. Assuming there are k different classes of features $f_1, f_2, \ldots, f_k$ one can build k matrices $S_1, S_2, \ldots, S_k$ corresponding to a similarity or proximity of objects with respect to each particular feature class. Depending on the nature of features, one can use different similarity measures, e.g. based on a Euclidian distance, cosine similarity or Jaccard Index. The final single representation S can then be obtained using an inclusive $S = \frac{1}{k} \sum_{i=1}^{k} S_i$ or exclusive $S = \prod_{i=1}^{k} S_i$ rule. The latter produces the sparsest result, which is beneficial from both computational and storage efficiency, however in our case decreased the overall quality of the model, and we went with the former.

We note that independently of the type of transformations described above *the effect of changing α and β (i.e. downvoting or upvoting off-diagonal elements of similarity matrices) is typically the most pronounced*, leading to a noticeable change in quality of recommendations.

6.2.3 Efficient computations

Matrix square root. Finding square root of an arbitrary matrix is a computationally intensive operation. However, by construction, matrices K and S are symmetric

[2] A sufficient (however not necessary) upper bound for the values of α and β can be estimated from the matrix *diagonal dominance* condition [66].

positive definite (SPD) and therefore can be represented in the *Cholesky decomposition* form: $S = L_S L_S^T$, $K = L_K L_K^T$, where L_S and L_K are lower triangular real matrices. This decomposition can be computed much more efficiently than the standard square root.

By a direct substitution it can be verified that SVD of the following auxiliary matrix

$$\widehat{A} \equiv L_K^T A_0 L_S = \widehat{U} \Sigma \widehat{V}^T, \tag{6.10}$$

also provides a solution to Eq. (6.6) and, therefore, it can be used to replace Eq. (6.7) with expensive square root computation. The connection between the auxiliary latent space and the original one in this case is given by (c.f. Eq. (6.8)):

$$\widehat{U} = L_K^T U, \quad \widehat{V} = L_S^T V. \tag{6.11}$$

Note that after singular vectors $\widehat{U}$ and $\widehat{V}$ are computed, there is no need to explicitly calculate inverses of L_S^T and L_K^T for finding U and V. It only requires to solve a corresponding triangular system of equations, which can be performed very efficiently [66].

Furthermore, matrices K and S are likely to be sparse for a broad set of real features and attributes. This can be also exploited via computation of the sparse Cholesky decomposition or, even better, incomplete Cholesky decomposition [66], additionally allowing to skip negligibly small similarity values. We note that for sparse similarity matrices the corresponding triangular part of their Cholesky factors also become sparse.

As an additional remark, Cholesky decomposition is fully deterministic and allows symbolic factorization to be used for finding the non-zero structure of its factors. This feature makes tuning HybridSVD more efficient: once the resulting sparsity pattern of the Cholesky factors is revealed it can be reused to speedup further calculations performed for different values of α and β due to Eq. (6.9), as it does not affect the sparsity structure.

Computing SVD. Note that there is also no need to directly compute the matrix product in Eq. (6.10), which would lead to a new potentially dense matrix. Instead, one can exploit the Lanczos procedure simialrly to the way it is used in PureSVD (Sec. 2.2.1). In order to find r principal singular vectors and corresponding singular values it is sufficient to simply provide a rule of how to multiply matrix product from

Eq. (6.10) by an arbitrary dense vector from the right and from the left. This can be implemented as a sequence of 3 matrix-vector multiplications. Hence, the added computational cost of the algorithm over the standard SVD is controlled by the complexity of multiplying Cholesky factors by a dense vector.

More specifically, given the number of non-zero elements nnz_A of the matrix A that corresponds to the number of the observed interactions, an overall computational complexity of the HybridSVD algorithm is estimated as $O(nnz_A \cdot r) + O((m+n) \cdot r^2) + O((J_K + J_S) \cdot r)$, where the first 2 terms correspond to PureSVD's complexity and the last term depends on the complexities J_K and J_S of multiplying triangular matrices L_K and L_S by a dense vector. In the scope of this work we are interested in the case when matrices K and S are sparse and therefore sparse Cholesky decomposition can be employed. Hence, J_K and J_S are determined by the corresponding number of non-zero elements nnz_{L_K} and nnz_{L_S} of the Cholesky factors. The total complexity in that case is $O(nnz_{tot} \cdot r) + O((m+n) \cdot r^2)$, where $nnz_{tot} = nnz_A + nnz_{L_K} + nnz_{L_S}$.

Generating recommendations. HybridSVD inherits the key properties of PureSVD, including simplified folding-in computation. Combining equations Eq. (6.7) and Eq. (6.10) and applying the folding-in technique gives the following very similar to Eq. (2.11) expression for the vector of predicted item scores:

$$\boldsymbol{r} \approx L_S^{-T} \widehat{V} \widehat{V}^T L_S^T \boldsymbol{a} = V_l V_r^T \boldsymbol{a}, \tag{6.12}$$

where $V_l = L_S^{-T} \widehat{V}$, $V_r = L_S \widehat{V}$ and $\boldsymbol{a}$ is a vector of user preferences. This can be applied to both known users and warm-start users. Here we assume that the matrix K is equal to identity (i.e. no side information about users is given). This corresponds to our experimental setup, described in Sec. 6.3.

6.3 Experiments

We conduct *three types of experiments*. The first experiment measures an *impact of data sparsity* on the performance of recommendation models. The key purpose of this experiment is to verify the main claims from Sec. 6.1 regarding the dependence on data sparsity. Another two experiments are *standard top-n recommendation scenario*

and *item cold start scenario*. Every experiment starts with a hyper-parameter tuning phase with $n = 10$ fixed and after the optimal parameter values are found they are used for final evaluation of recommendations quality.

6.3.1 Evaluation methodology

In the *sparsity test* experiment we sequentially take 1, 3, 10, 30 and 100% of interaction data to vary its density and build recommendation models on top of it. We preliminarily exclude all known preferences of a set of randomly sampled test users. We additionally exclude any test user preferences that are not present in all data subsamples simultaneously. This ensures a fair and consistent comparison. For each test user we holdout a single item at random from his or her preferences. The rest of the items are used to generate recommendations which are then evaluated against the holdout.

In the *standard scenario* we consequently mark every 20% of users for test. Each 20% partition always contains only those users who have not been tested yet. We randomly withdraw a single item from every test user and form a holdout set based on these items. After that the test users are merged back with the remaining 80% of users and form a training set. During the evaluation phase we generate a ranked list of top-n recommendations for every test user based on their known preferences and evaluate it against the holdout.

In the *cold start scenario* we perform 80%/20% partitioning of the list of all unique items. We select items from a 20% partition and mark them as cold start items. Users with at least one cold start item in the preferences are marked as the test users. Users with no items in their preferences, other than cold start items, are filtered out. The remaining users form a training set with all cold start items excluded. Evaluation of models in that case is performed as follows: for every cold start item we generate a ranked list of the most pertinent users and evaluate it against one of the test users chosen randomly among those who have actually interacted with the item.

In both *sparsity test* and *standard scenario* we try to predict which items will be the most relevant for a set of selected test users. Alternatively, in the *cold start scenario* we try to find those users who are likely to be the most interested in a set of selected cold start items. In both *standard* and *cold start* experiments we perform

a 5-fold cross validation and average the results over all 5 folds. We also report 95% confidence intervals based on the paired t-test criteria.

The quality of recommendations is measured with the help of *hit-rate* (HR) and *average reciprocal hit-rank* (ARHR) metrics [43]. In our settings with a single holdout entity the ARHR metric is equivalent to *mean reciprocal rank* (MRR). The resulting evaluation scores computed for different values of n (from 1 to 30) are denoted as MRR@n or HR@n. We also use the MRR score as a selection criterion during the hyper-parameter tuning phase.

6.3.2 Datasets

We have used *MovieLens-10M* (ML10M), *MovieLens-1M* (ML1M) and *BookCrossing* (BX), datasets hosted by Grouplens[3]. These datasets provide snapshots of real users' behavior and are widely used in a research literature for benchmarking recommendation algorithms. Beyond that, we choose these particular datasets due to their substantial differences in an internal data structure. ML1M dataset contains very active users with a lot of feedback provided for various items. Conversely, interaction data in the BX dataset is very scarce as users tend to provide their feedback to a considerably fewer number of items comparing to the full assortment. ML10M is very similar to ML1M, however its size is sufficient for reliable subsampling of data and performing gradual transition from high to low sparsity levels.

These datasets allow us to assess whether the resulting sparsity of the data affects the importance of side information in terms of recommendations quality. As has been noted, the hypothesis behind this assessment is that the lack of collaborative information makes it more difficult to reveal intrinsic relations within the data without any side knowledge. In contrast, a sufficient amount of collaborative information may totally hinder the positive effect of side knowledge. Moreover, if chosen side features do not play a significant role in a user decision-making process, recommendation models may suffer from learning non-representative relations.

As we are not interested in the rating prediction, the settings with only binary feedback are considered in our experiments. In the case of the BX dataset we select only the part with an implicit data. In these settings a recommendation model predicts

[3]https://grouplens.org/datasets/

how likely is a user to interact with a recommended book. We additionally preprocess the data by filtering out users with more than 2000 or less than 3 items in their preferences. Items with only one interaction are also removed. This resulted in the dataset with 15936 users, 87068 items and 0.033% density. The information about authors and publishers available in the dataset is used to build side similarity matrices. We employed simple cosine similarity measure for that purpose.

Both Movielens datasets are binarized with a threshold value of 4: lower ratings are filtered and the remaining ratings are set to 1. With this setup in the standard scenario a recommendation model predicts how likely is a user to rate a recommended movie with 4 or 5 stars. As the result, ML1M consists of 6038 users, 3532 items and has a 2,7% density, while ML10M has 69797 users, 10255 items and 0.7% density.

The Movielens datasets contain only genres information. We have crawled the TMDB database[4] to additionally extract *cast, directors and writers* information. As the lists of cast and directors are meaningfully ordered (e.g. movie actors are sorted according to the importance of their role) we employed Weighted (a.k.a. generalized) Jaccard Index [34]. It allows to compare sets with respect to the weights associated with set elements and in our case the weights are obtained as reciprocal ranks of actors and directors respectively. For other features with used cosine similarity with standard row normalization.

Due to a high number of movies with Drama listed as one of their genres, the resulting density of similarity matrices for Movielens datasets was around 50%, which can already be considered a dense matrix. One simple way to reduce the density is to remove that genre entirely as uninformative or leave it only for a smaller subset of items (e.g., for long-tail items). We, however, proceeded as is. Even in that case computing Cholesky decomposition took less than 10s on a laptop for ML10M.

6.3.3 Baseline algorithms

We compare the proposed *HybridSVD* model to several standard baseline models, including *PureSVD*. We also provide comparison with *Factorization Machines* (FM) [132] as one of the most popular models, used to win several recommendation challenges in the past. FM allows to easily incorporate any sort of side information in the

[4]https://www.themoviedb.org

form of sparse one-hot encoded vectors. Below is the description of implementation details for each model:

- *CB* is a hybrid approach based on an aggregation of similarity scores (content-based information) computed with the help of known user preferences (collaborative information). In the *standard scenario* the aggregated *item scores* are $\boldsymbol{r} = S\boldsymbol{a}$. It is used to directly order items by their similarity to a test user's preferences, encoded by a sparse binary vector $\boldsymbol{a}$. In turn, in the *cold start scenario* we calculate the aggregated *user scores* $\bar{\boldsymbol{r}} = A_0\boldsymbol{c}$, where $\boldsymbol{c}$ denotes the similarity vector of a cold start item to other items. The resulting vector of scores $\bar{\boldsymbol{r}}$ represents how pertinent each user is to the cold start item. *This vector is also used for the SVD-based models as a replacement of known user preferences in the cold start regime* (see below).

- *PureSVD* is the model described in Sec. 2.2.1. The model is not directly applicable in the cold start regime, as there is no preference information available.

- *FM* is a Factorization Machines model with ranking optimization objective used instead of a standard one. We use implementation from Graphlab Create software package[5]. The model uses general formulation with user and item biases and incorporates it into a binary prediction objective based on a sigmoid function. The optimization task is performed by SGD with adaptive learning rate. Note that in the case of implicit feedback the interaction matrix becomes complete (even though sparse), which would make the SGD-based optimization infeasible. However, instead of learning over all data points, the algorithm employs a negative sampling technique. It learns over all positive examples (rated items) and a fixed pre-defined number of negative examples (unrated items) sampled randomly[6].

- *MP* model recommends top-n the most popular items (in the standard scenario) or the most active users with the highest overall number of preferences (in the cold start scenario).

[5]https://turi.com/download/install-graphlab-create.html

[6]as defined in the *RankingFactorizationRecommender* class from the GraphLab documentation at https://turi.com/

- *RND* model generates recommendations based on random item/user selection in standard/cold start scenarios.

Recall that in the cold start scenario we try to recommend known users to cold start items. Hence, the preference data is not available and the folding-in approach is not directly applicable. To alleviate the problem we take an output of the CB model $\bar{\boldsymbol{r}}$ as a preference vector of a cold start item against all known users. Then, for every cold start item we can generate prediction scores as $\boldsymbol{r} \approx UU^T\bar{\boldsymbol{r}}$, where matrix U is computed by either PureSVD or HybridSVD. To explicitly denote this change we mark PureSVD as *PureSVD+CB*. We do not add *CB* to HybridSVD name to avoid visual cluttering.

6.3.4 Hyper-parameters tuning

We assess the quality of algorithms in terms of MRR@10 and HR@10 with the main focus on the MRR metric. We note that in our experiments the performance demonstrated by algorithms in terms of the HR metric is highly correlated with the performance in terms of MRR. However, we used HR scores to monitor the generalization of algorithms. For example, during the model tuning phase in the FM case some sets of hyper-parameters could provide high values of MRR and considerably lower values of HR comparing to other sets. In order to avoid such overfitting, we shifted the selection of hyper-parameters towards slightly lower MRR but reasonably high HR.

We test all factorization models on a wide range of rank values (i.e. a number of latent features). The HybridSVD model is also evaluated for 3 different values of α: 0.1, 0.5 and 0.999. Similarly to the standard SVD case (see Sec. 2.5.2) and unlike the majority of MF methods, once the model is computed for some rank r_{max} with a fixed value of α, *we immediately get an access to all the models with a lower rank* $r < r_{max}$ *by a simple rank truncation.* In other words, in order to obtain a rank-r model of HybridSVD it only requires to select the first r principal components of the model of rank r_{max} without any additional optimization. This significantly simplifies the hyper-parameter tuning procedure as it eliminates the need for expensive model recomputation during the parameter grid search.

Configuration of the FM model consists of the following hyper-parameters: *regularization coefficients* for the bias terms, interaction terms and ranking (negative sampling) terms, *initial SGD step size*, the *number of negative samples* and the *number of epochs*. In our experiments simpler SGD optimization was performing slightly better than ADAGRAD [47].

Note that hyper-parameter space of the FM model quickly becomes infeasible with the increased granularity of a parameter grid. Not only this model requires more parameters to tune, we also do not have the luxury of a simplified rank optimization as in the case of HybridSVD. The problem is magnified by significantly longer training times in the case of FM. For example, on the ML1M dataset the FM model of rank 50 requires about 300s to converge (16-core Intel Xeon CPU E5-2640 v2 @2.00GHz), while HybridSVD takes only about 10s and PureSVD takes less than 1s.

In order to deal with this issue we employ a *Random Search* strategy [16] and limit the number of possible hyper-parameters combinations to 120. We additionally perform an extensive grid-search in the closest proximity of the hyper-parameters found during the Random Search phase. This allows to quickly test for more optimal values that could be missed due to randomization. The tuning is always performed on a single fold of cross-validation by additionally splitting it into train and validation sets. The parameters, found during this step are then fixed for all folds.

6.4 Results and discussion

Results for standard and cold start scenarios are depicted in Fig. 6.3 and Fig. 6.4. We report confidence intervals only for the final top-n recommendation results (bottom rows). Confidence regions for the rank estimation experiments (top rows) are not reported for the sake of picture clarity.

As can be seen, HybridSVD models exhibit very different behavior on the two datasets. For highly sparse BX data, where the number of known preferences per user is much lower than in the ML1M case, even a simple information such as book author helps HybridSVD to learn a better representation of behavioral patterns, which is reflected by a generally higher quality of recommendations. The difference is more pronounced in the standard scenario (see the right column in Fig. 6.3) than in the cold start settings.

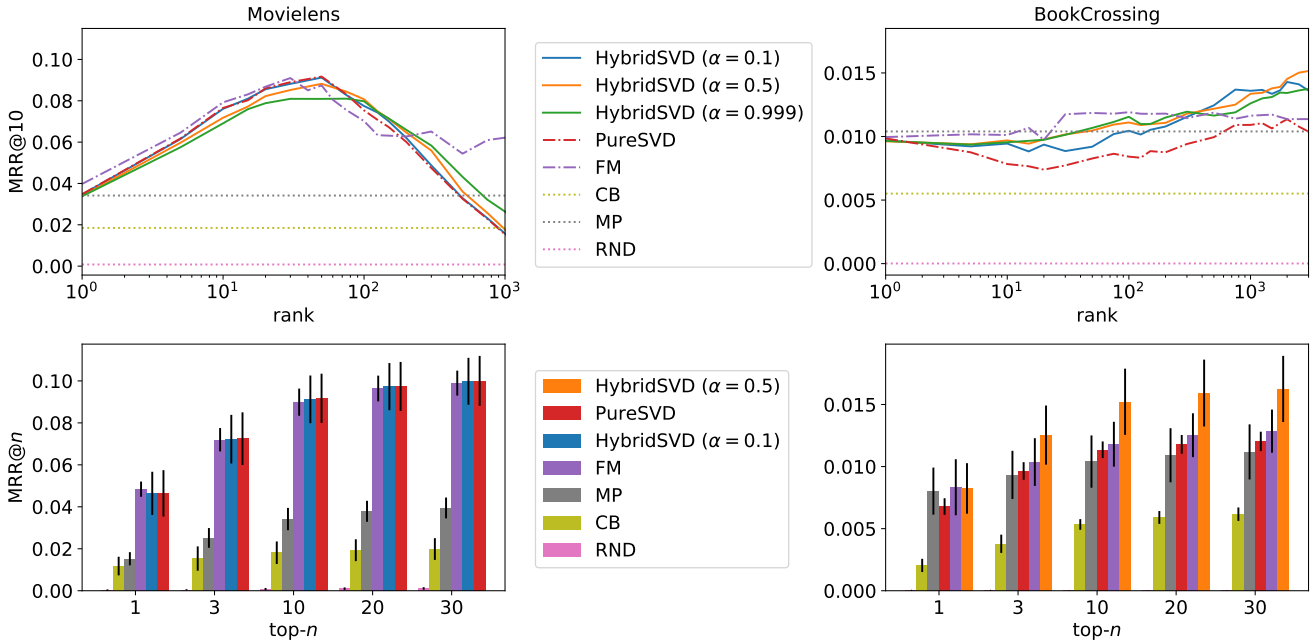

Figure 6.3: Experimentation results for standard scenario. The left column corresponds to Movielens-1M, the right column - to BookCrossing. The first row represents rank estimation experiments, the second row - final evaluation of top-n recommendations quality. The confidence intervals are reported as black vertical lines on top of the bars.

6.4.1 Standard scenario

Remarkably, the highest MRR score in the BX case, achieved by PureSVD at rank 2000 in standard scenario, can be achieved with HybridSVD ($\alpha = 0.5$) at rank 100. Moreover, unlike PureSVD, the score of some HybridSVD models exhibits a positive growth rate even at the rank 3200, at which we simply stopped our experiments. This means that potentially even higher evaluation scores can be achieved (leaving aside the practical aspect of huge rank values).

It should be noted that FM model also performs well on BX data in standard settings and achieves the best PureSVD score at the lowest among other models value of rank (around 30). However its maximum MRR score is much lower than the maximum score achieved by HybridSVD (see bottom-right graph of Fig. 6.3). The quality of the FM model also seems to be less sensitive to the rank value, when other hyperparameters are optimally tuned. This is indicated by several almost flat regions on the rank estimation curves (top row).

Table 6.2: HybridSVD is more stable and reliable when the data sparsity is increasing. Reported numbers are the MRR@10 scores, obtained on the Movielens-10M dataset.

Fraction of data	1%	3%	10%	30%	100%
Resulting density	0.04%	0.05%	0.10%	0.23%	0.70%
HybridSVD	**0.045**	**0.049**	**0.056**	**0.077**	0.105
PureSVD	0.037	0.038	0.054	0.076	**0.112**
MP	0.039	0.045	0.042	0.044	0.042

In the ML1M case we were unable to outperform PureSVD in standard scenario (the left column in Fig. 6.3) and almost all factorization models achieve similar scores. The FM model requires slightly lower ranks to achieve the comparable quality in that case. Interesting to note that HybridSVD with the highest value of α equal to 0.999 underperforms other factorization models. All this suggests that relying too much on side information confuses the model in that case.

This observation resonates well with the results in [123]. As the authors argue, "*even a few ratings are more valuable than metadata*". Indeed, on the relatively *dense* movie ratings data additional features such as movie genres or actors seem to bring not enough new knowledge into an understanding of common patterns and probably lead to overspecialization of models.

In contrast, in the case of very *sparse* BX data higher contribution of item features (i.e. higher values of α) lead to a generally better quality of recommendations, which indicates that *without side knowledge factorization models are unable to reliably recover hidden relations* and that using only the collaborative information in this case may be insufficient.

This result is also suported by our *sparsity test* experiment on the ML10M dataset (see Table 6.2). As can bee seen, while PureSVD achieves the highest score on full data, its quality quickly decreases as less information about user preferences is given. At extreme sparsity levels it even falls below the quality of non-personalized popularity-based model. In contrast, HybridSVD exhibits more reliable behavior and handles extreme sparsity much better.

With the help of HybridSVD we demonstrate that side knowledge allows to create additional "virtual" connections between related entities, which in turn helps to alleviate the lack of preferences data. Inability to account for such information in

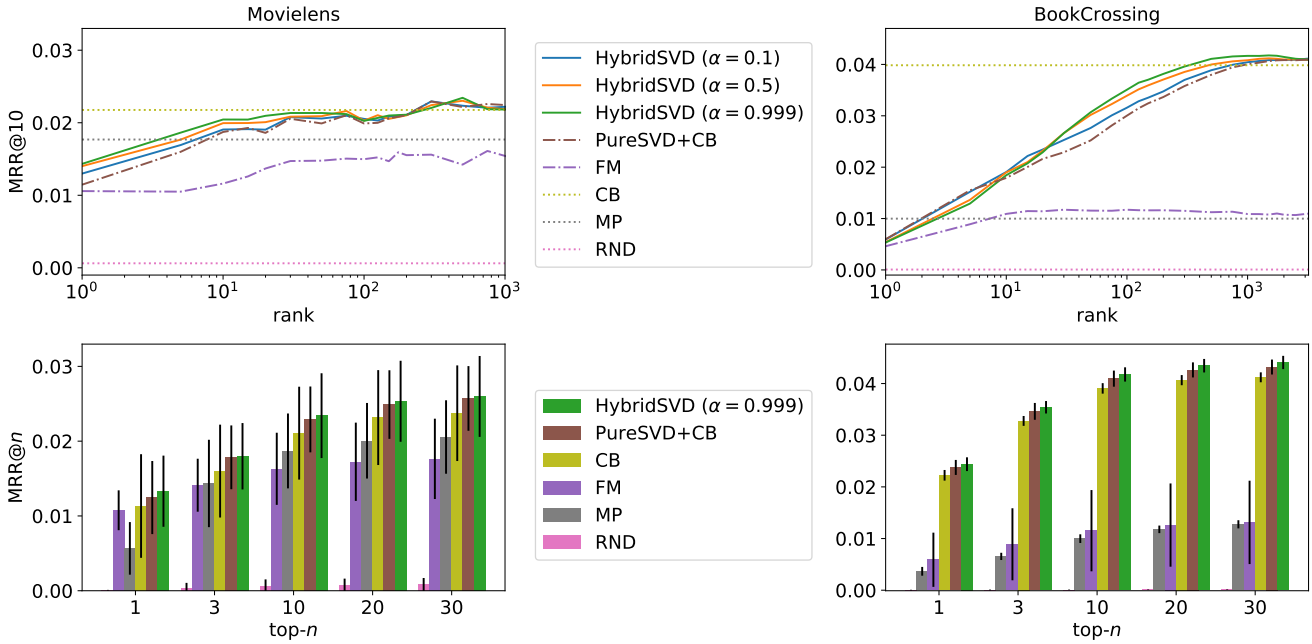

Figure 6.4: Experimentation results for item cold start scenario. The left column corresponds to Movielens, the right column - to BookCrossing. The first row represents rank estimation experiments, the second row - final evaluation of top-n recommendations quality. The confidence intervals are reported as black vertical lines on top of the bars.

the PureSVD approach leads to its high sensitivity to the sparsity structure of an input data. This result addresses (at least partially) the question from the end of Sec. 6.1.1.

6.4.2 Cold start scenario

In the cold start settings HybridSVD consistently outperforms the FM model sometimes by a significant margin (see Figure 6.4). A possible reason is that HybridSVD uses more data to generate recommendations. It utilizes the information about similarity of items based on their features, while the FM model directly relies on the latent representations of the features, when no preferences data is available. This puts the models into a sort of unfair comparison.

One possible way to avoid that is to perform the folding-in optimization in the FM model and try to fit the similarity data instead of interactions. Such incremental updates could potentially improve the quality of the model. However, this is not as straightforward as a simple matrix-vector multiplication provided by HybridSVD and requires additional model modifications.

Another common observation is that the CB approach, which relies on a simple heuristic, performs remarkably well comparing to more sophisticated models. Even though it is formally outperformed by the HybridSVD approach, the difference between them is negligibly small. Moreover, even the PureSVD+CB model behaves comparably to HybridSVD, except that *the rank of HybridSVD to achieve the same quality is 5 to 10*x *smaller* (mind the logarithmic scale for rank values).

The performance of the HybridSVD approach is consistent, favoring the higher values of α. Unsurprisingly, in the cold start regime it relies a lot on side information for both datasets. Generally, the proposed approach provides a flexible tool to control the contribution of side features into the model's predictions. It allows to adjust recommendations based on the meaningfulness of side information. Moreover, it allows to enforce the desired latent feature space structure as in the example with genres in Fig. 6.2.

6.5 Conclusions and further research

We have generalized PureSVD approach to support user and item side information. The model allows to saturate collaborative data with additional feature-based relations and in certain cases improve the quality of recommendations. The model seems to be especially suitable for the data with scarce user activity when the number of observed user preferences is low. In a "saturated" environment with a high amount of user feedback the model seems to provide no benefit over PureSVD. We have also proposed an efficient computation scheme for both model construction and recommendation generation in online settings.

Despite being a flexible instrument for adjusting the contribution of side information into the final prediction quality, the pre-processing step of HybridSVD requires some amount of efforts. Finding a way to avoid an explicit construction of side similarity matrices seems to be an interesting direction for further research.

Chapter 7

Higher order hybrid preference model

Up until now, we have proposed two models that address different aspects of the limited preference information problem. The CoFFee model, introduced in Chap. 5, tackles the problem of a proper feedback representation, which fits nicely into a tensor-based formulation. The main benefit of this representation is that it helps to account for additional commonalities in user behavior and improve the quality of recommendations without the need for any additional data. Its advantage over conventional models becomes especially vivid in the "almost" cold start case, when known user preferences consist of only one or a few items.

On the other hand, HybridSVD approach, introduced in Chap. 6, makes use of additional data sources, not related to actual user preferences. For this purpose, it employs a generalized formulation of SVD and enriches standard SVD-based models with side information about users and items. Based on that information, the model measures how similar users or items are and virtually links them within collaborative data. This allows uncovering more valuable patterns, which would otherwise stay unrecognized. Moreover, it helps to battle extreme data sparsity.

Clearly, the fundamental ideas behind the proposed models are complementary, which raises the question of whether it is possible to integrate one model into another. In this chapter, we address this question by presenting a new model that directly combines our previous models within a unified hybrid tensor-based approach. It allows to represent user preferences adequately and at the same time leverages side information in order to improve recommendations' quality and handle data sparsity. We provide efficient computational schemes for both offline learning and online recommendation generation in dynamic environments.

7.1 Motivation for a joint model

In the experiments with HybridSVD, we have only considered binary feedback data. It is a reasonable formulation when user-item interactions have the simplest form of an implicit feedback (e.g., likes or purchases). However, in general, user feedback may have a more complex nature and often embodies several distinct types or modalities, which require careful treatment.

For example, implicit feedback may split into different types of actions, such as click on a product page, placing an order or actual product purchase. Evidently, this corresponds to different levels of user engagement. Assigning appropriate weights to these actions in order to generate a single number (i.e., relevance score) used in the matrix-based formulations is a challenging empirical task.

Similar reasoning applies to explicit feedback as well. We have already touched this problem in Chap. 5. In addition to what was discussed there, we find it necessary to provide another intuitive example of a common feedback representation fallacy with more general implications. Consider a user who has assigned a 5-star rating to one movie and gave only 2 stars to another. From here it does not immediately follow that the user admires the former movie exactly 2.5 times higher than the latter. It only implies that the user prefers one movie to another. *This difference cannot be expressed with simple arithmetic rules* and should be treated in terms of an ordinal nature of feedback.

There is an even more substantial problem particularly related to the HybridSVD formulation, which can be illustrated with the following example. If user Alice rates “Scarface” movie with 2 stars (negative preference) and user Bob rates “Godfather” with 5 stars (positive preference), then even though these two movies are quite similar in terms of a genre, it is unlikely that Alice and Bob have similar tastes. However, this is not what the HybridSVD model will actually learn, as it will create an additional link between users based on movie genre similarity and users will become closer to each other in the latent feature space. In our combined model we aim to resolve that problem as well by separating rating values in the third dimension.

From the CoFFee model perspective, side information not only serves the purpose of generating a more meaningful latent representation of users and items but also helps to address one of the key challenges of the tensor-based formulation – an

increased data sparsity. Indeed adding new dimensions without providing more data inevitably reduces the density. This, in turn, may have a substantial negative impact on the generalization ability and the quality of the model.

7.2 Proposed approach

Following the same way SVD is generalized by Tucker decomposition, an auxiliary matrix from Eq. (6.10) can be generalized by an auxiliary tensor $\widehat{\mathcal{A}}$:

$$\widehat{\mathcal{A}} \equiv \mathcal{A} \times_1 L_K^T \times_2 L_S^T \times_3 L_R^T,$$

where L_R is a Cholesky factor of some SPD similarity matrix R that corresponds to the feedback dimension. With this formulation the model allows to naturally handle cases described in Sec. 7.1 by *linking only items with the same feedback value.* This is achieved by setting $R = I$. The model, however, provides much more flexibility and allows to go beyond that scenario. In the presence of feedback similarity/correlation data (i.e. when R is not just the identity matrix), *the model allows to diffuse connections across feedback dimension* when it is required by the task or dictated by the structure of feedback data, e.g. when some feedback values are "closer" to each other in some sense. We will leave the discussion of its meaning for the later (see Section 7.6).

The recommendation model is obtained from a low rank approximation of $\widehat{\mathcal{A}}$. As in the CoFFee case, it can be achieved with the help of TD:

$$\widehat{\mathcal{A}} \approx \mathcal{G} \times_1 \widehat{U} \times_2 \widehat{V} \times_3 \widehat{W}, \tag{7.1}$$

where factor matrices are also required to have orthonormal columns. We call this model *HybridCoFFee* to emphasize its ability to adequately represent higher order preference data and saturate it with side information.

Note that factor matrices $\widehat{U} \in \mathbb{R}^{M \times r_1}$, $\widehat{V} \in \mathbb{R}^{N \times r_2}$ and $\widehat{W} \in \mathbb{R}^{F \times r_3}$ are defined in an auxiliary latent space. The latent representation of users, items and feedback in the original space is then given by

$$U = L_K^{-T} \widehat{U}, \quad V = L_S^{-T} \widehat{V}, \quad W = L_R^{-T} \widehat{W}. \tag{7.2}$$

Columns of the resulting factor matrices satisfy K-, S- and R-orthogonality property, i.e. $U^T K U = I_{r_1}$, $V^T S V = I_{r_2}$ and $W^T R W = I_{r_3}$ (I_r is an identity matrix of size r).

Similarly to the HybridSVD case, this imposes an additional constraint that *structures the latent feature space according to real characteristics of the modelled entities.*

We also add a control of an overall contribution of side information into the learned latent representation by representing the similarity matrices in the form $K = I + \alpha K_0$, $S = I + \beta S_0$ and $R = I + \gamma R_0$, where zero-diagonal matrices K_0, S_0 and R_0 actually encode side information-based relations and α, β, γ are non-negative weighting parameters. Obviously, by setting α, β, γ to zero one gets the standard CoFFee model.

Despite its similar look, the model, however, has a few substantial differences from the standard TD that require careful handling. In the next section we show how to efficiently compute it by a corresponding modification of the optimization objective.

7.3 Efficient computations

As in the CoFFee model case, a low rank approximation, defined by Eq. (7.1), can be obtained with a HOOI algorithm. It solves the corresponding least squares problem by an alternating optimization procedure, where the objective is minimized with respect to one of the latent feature matrices while the other two are fixed. As shown by the authors of HOOI, the problem conveniently reduces to the following maximization task:

$$\max_X \|\widehat{\mathcal{A}} \times_1 \widehat{U}^T \times_2 \widehat{V}^T \times_3 \widehat{W}^T\|_{\mathrm{F}}^2, \tag{7.3}$$

where X is picked iteratively from $\left\{\widehat{U}, \widehat{V}, \widehat{W}\right\}$ at each alternating optimization step. The task can be efficiently solved by the means of SVD (see Alg. (2)).

7.3.1 Hybrid tensor factorization

Note, however, that unlike the preference tensor in the CoFFee case, tensor $\widehat{\mathcal{A}}$ *is not necessarily sparse* and computing it quickly becomes *the main bottleneck in terms of system resources usage* with the growth of the problem size. In order to avoid its explicit formation we rewrite the inner term of Eq. (7.3) as

$$\widehat{\mathcal{A}} \times_1 \widehat{U}^T \times_2 \widehat{V}^T \times_3 \widehat{W}^T \equiv \mathcal{A} \times_1 U_K^T \times_2 V_S^T \times_3 W_R^T, \tag{7.4}$$

Algorithm 2: Practical algorithm for hybrid HOOI

Input : Tensor $\mathcal{A}$ in sparse COO format,
Tensor decomposition ranks r_1, r_2, r_3,
Cholesky factors L_K, L_S, L_R

Output: $\mathcal{G}, \widehat{U}, \widehat{V}, \widehat{W}$

Initialize $\widehat{V}, \widehat{W}$ by random matrices with orthonormal columns.

Compute $V_S = L_S \widehat{V}, W_R = L_R \widehat{W}$.

repeat

$\widehat{U} \leftarrow r_1$ leading left singular vectors of $L_K^T A^{(1)} (W_R \otimes V_S)$

$U_K \leftarrow L_K \widehat{U}$

$\widehat{V} \leftarrow r_2$ leading left singular vectors of $L_S^T A^{(2)} (W_R \otimes U_K)$

$V_S \leftarrow L_S \widehat{V}$

$\widehat{W}, \Sigma, Z \leftarrow r_3$ leading singular triplets of $L_R^T A^{(3)} (V_S \otimes U_K)$

$W_R \leftarrow L_R \widehat{W}$

$\mathcal{G} \leftarrow$ reshape matrix ΣZ^T into shape (r_3, r_1, r_2) and transpose

until *norm of the core ceases to grow or exceeds maximum iterations*;

where we use the substitution $U_K = L_K \widehat{U}$, $V_S = L_S \widehat{V}$, $W_R = L_R \widehat{W}$ and utilize the multiplication properties of a series of matrices in the n-mode product.

With the latter representation in Eq. (7.4) one can follow a standard technique to separate any factor matrix from the other two in order to perform an alternating optimization step. This is achieved by the virtue of tensor unfolding defined in Sec. 3.1.1 and with the help of an n-mode product properties. For example, to optimize for $\widehat{U}$ one arrives at the following expression after combining Eq. (7.3) and Eq. (7.4) with the aforementioned properties:

$$\max_{\widehat{U}} \|\widehat{U}^T L_K^T A^{(1)} (W_R \otimes V_S)\|_{\mathrm{F}}^2,$$

where matrix $A^{(i)}$ *denotes mode-i unfolding of* $\mathcal{A}$ and $\otimes$ stands for Kronecker product. The corresponding solution is then given by the leading left singular vectors of $L_K^T A^{(1)} (W_R \otimes V_S)$. Similar transformations along modes 2 and 3 give the update rules for $\widehat{V}$ and $\widehat{W}$ respectively. See Alg. (2) for full description of the optimization process.

Note that the product $A^{(1)} (W_R \otimes V_S)$ has the same structure as in the standard TD case. Therefore, for moderately sized problems it can be computed without ex-

plicit construction of $W_R \otimes V_S$ by performing a series of matrix multiplications with a corresponding tensor unfolding [9]. For larger problems the memory bottleneck induced by intermediate computation results can be circumvented by iteratively updating entries of the final result in a simple nested loop instead of performing matrix multiplications.

7.3.2 Online recommendations

As in the case with CoFFee or HybridSVD the orthogonality of columns in factor matrices allows to derive an efficient expression for higher-order hybrid folding-in. In the user case, it helps to solve the problem of recommendations for unrecognized or newly introduced users with only a few known preferences. Likewise, in the item case it allows to quickly find an item's representation in the latent feature space based on a few interactions with it. As an example, the following expression is a generalization of the tensor folding-in to the hybrid case, which allows to estimate new user preferences (c.f. Eq. (5.9)):

$$\bar{P} = V V_S^T P W_R W^T, \tag{7.5}$$

where V and W are defined according to Eq. (7.2). This allows to avoid recomputing the whole model in response to frequent system updates. As has been noted in the previous chapters, it is especially viable in highly dynamic online environments, where users expect an instant response from recommendation services or where new items arrive rapidly. In our experiments *we use this formula to generate recommendations for the known users as well*, not only newcomers.

7.3.3 Rank truncation

Hyper parameter-tuning can be a tedious task. As we have already mentioned in Sec. 2.5.2, unlike many other approaches SVD-based methods provide a luxury of minimal hyper parameter tuning via simple rank truncation of latent factors. Even though it is not directly applicable in the tensor case, it is still possible to avoid redundant computation of the model with lower multilinear rank values by the means of *tensor rounding* technique. More formally, given some factor matrix $X \in \{\widehat{U}, \widehat{V}, \widehat{W}\}$, which corresponds to some mode $i \in \{1, 2, 3\}$, and a new rank value $r < \text{rank}(X)$, the first step is to compute r leading singular triplets U_r, Σ_r, V_r of the unfolded core $G^{(i)}$.

Then the new factor matrix X_r of the reduced rank r is calculated as $X_r = XU_r$ and the new truncated core $\mathcal{G}_r$ is obtained by reshaping matrix $\Sigma_r V_r^T$ back into the tensor of order 3 with the conforming dimensionality. Note that due to typically small multilinear rank values, finding SVD of an unfolded core is computationally cheap and an overall procedure is very efficient.

7.4 Evaluation methodology

We conduct a 5-fold cross-validation (CV) experiment for standard top-n recommendation scenario by performing splits by users. At every fold we randomly mark 20% of users that were not yet tested. We *randomly* hide 10 consumed items of every marked user to form the *holdout set*. This allows to have both high and low ratings in the holdout and, therefore, to *evaluate recommendations against both negative and positive user preferences*. User feedback is considered to be positive if the rating value is equal or above 4 (including 4.5 if it is present in data) with the highest rating being 5. The remaining items from the marked users as well as all the preferences of 80% of unmarked users form the *training set*. At each fold we generate recommendations for the marked users and evaluate them against the holdout. CV results are averaged and reported along with 95% confidence intervals based on the paired t-test criterion. We also repeat the sparsity test experiment described in Sec. 6.3.1. This time we do not binarize ratings. No cross-validation is available in this case due to a fixed testset.

Metrics. As has been shown in Chap. 5, standard evaluation metrics exhibit a *positivity bias*, i.e. only consider the performance in terms of how relevant recommended items are and completely disregard how likely it is to get recommended something irrelevant. The latter, however, may have a dramatic impact on the perceived quality of a recommendation service and affects user retention. In order to account for such effects we follow the evaluation scheme introduced in Sec. 5.3 and in addition to the standard relevance- and ranking-based metrics also report performance of models against the nDCL score. As we have shown, the latter serves as a proxy measure for user disappointment and estimates how likely is a user to remain unsatisfied with provided recommendations. We note that *models with similar nDCG may have different nDCL score*.

Datasets. We use the same benchmark datasets as in previous chapters: *MovieLens-1M* (ML1M), *MovieLens-10M* (ML10M), and *BookCrossing* (BX). These datasets have very different levels of data sparsity and therefore allow to examine how sensitive our model is to the lack of collaborative information in comparison to other models. In addition to that we randomly sample 3% of ratings from the ML10M dataset to conduct the sparsity test experiment. We do not perform any special preprocessing for the Movielens datasets. In the BX case we filter out users with more than 1000 ratings as they are unlikely to represent real consumption patterns. We also remove books with only one rating provided by a single user as unreliable. Ratings in the BX dataset range from 1 to 10. In order to have uniform representation across all datasets, we divide them by 2, giving a range from 0.5 to 5 with 0.5 step, similarly to ML10M. Ratings in the ML1M dataset are integer values from 1 to 5.

Algorithms. We compare our method to both CoFFee and HybridSVD approaches. We additionally use standard baseline models, namely PureSVD, WRMF (as in Chap. 5), a heuristic model that recommends items based on their aggregated similarity to known user preferences (CB), and a non-personalized model that simply recommends the most popular items (MP). Models are tuned on the first CV fold and the best found configuration corresponding to the *highest nDCG score* is then used across the remaining folds. In the case of PureSVD the only varying hyper-parameter is the number of latent factors. For WRMF tuning we perform Random Search on the hyper-parameter grid by sampling 60 points, corresponding to different combinations of weighting function parameters (according to Eq. (2.31)), regularization coefficients and rank values. In the CoFFee model we tune its multilinear rank with the requirement for mode-1 and mode-2 ranks to be always equal and take values from the same range as the rank of PureSVD. Mode-3 rank takes values from {2, 3, 4}. In the HybridSVD case we firstly tune its rank with a fixed weight value for side information set to 0.5. After an optimal rank is found we perform additional evaluation to find the most suitable weight value from {0.1, 0.5, 0.9}. Similar procedure is performed for HybridCoFFee with the same requirement on rank values as for the CoFFee model. SVD-based models use rank truncation to avoid redundant calculations during rank tuning. Likewise, tensor-based models use tensor rounding described in Sec. 7.3.3.

Side information. As in the HybridSVD case we used the information from TMDB database[1] to complete movie data in the Movielens datasets with information about *cast, directors* and *writers* along with already present *genre* information. BX dataset provides additional information about *authors* and *publishers*. There is no additional information about users or ratings, which renders L_K and L_R to be simply identity matrices. For each dataset we *inclusively merge all side data* by independently constructing similarity matrices S_i for each particular feature i and then combining them into a single similarity matrix with a simple summation $S = \frac{1}{n_f}\sum_{i=1}^{n_f} S_i$, where $n_f = 4$ in the Movielens case and $n_f = 2$ in the BX case. Accordingly, we used the same similarity measures for constructing S_i as in the HybridSVD case.

7.5 Results

We report 3 key evaluation metrics for all three datasets, which allow to assess the quality of recommendation models: an overall ratio of relevant recommendations to irrelevant, measured by Reciever Operator Characteristic curve (ROC), position of *relevant* predictions in top-n recommendation list, measured by nDCG and position of *irrelevant* predictions in top-n recommendation list measured by nDCL (see Fig. 7.1). Note, that there is typically some balance between high relevance of recommendations and high probability to generate irrelevant recommendations as well.

In order to correctly interpret results it is important to note, that low nDCG scores do not necessarily mean low quality of recommendations. If a model with low nDCG produces high enough ROC curve and at the same time shows low nDCL it simply means that the model makes more "safe" recommendations. Instead of recommending something irrelevant it pushes to the top more of unrated items, which is generally a better strategy. In contrast, if the relevance-based scores as well as nDCL score are all low, it indicates a poor performance.

For example, as can be seen from the first row of Fig. 7.1, both CB and MP models have low nDCL; however, their relevance-based scores are also low, which means that these models provide unsatisfactory recommendations. In contrast, HybridSVD provides the highest (or one of the highest) nDCG score in general. However, it also pushes one of the highest numbers of irrelevant items to the top of recommendations

[1] https://www.themoviedb.org

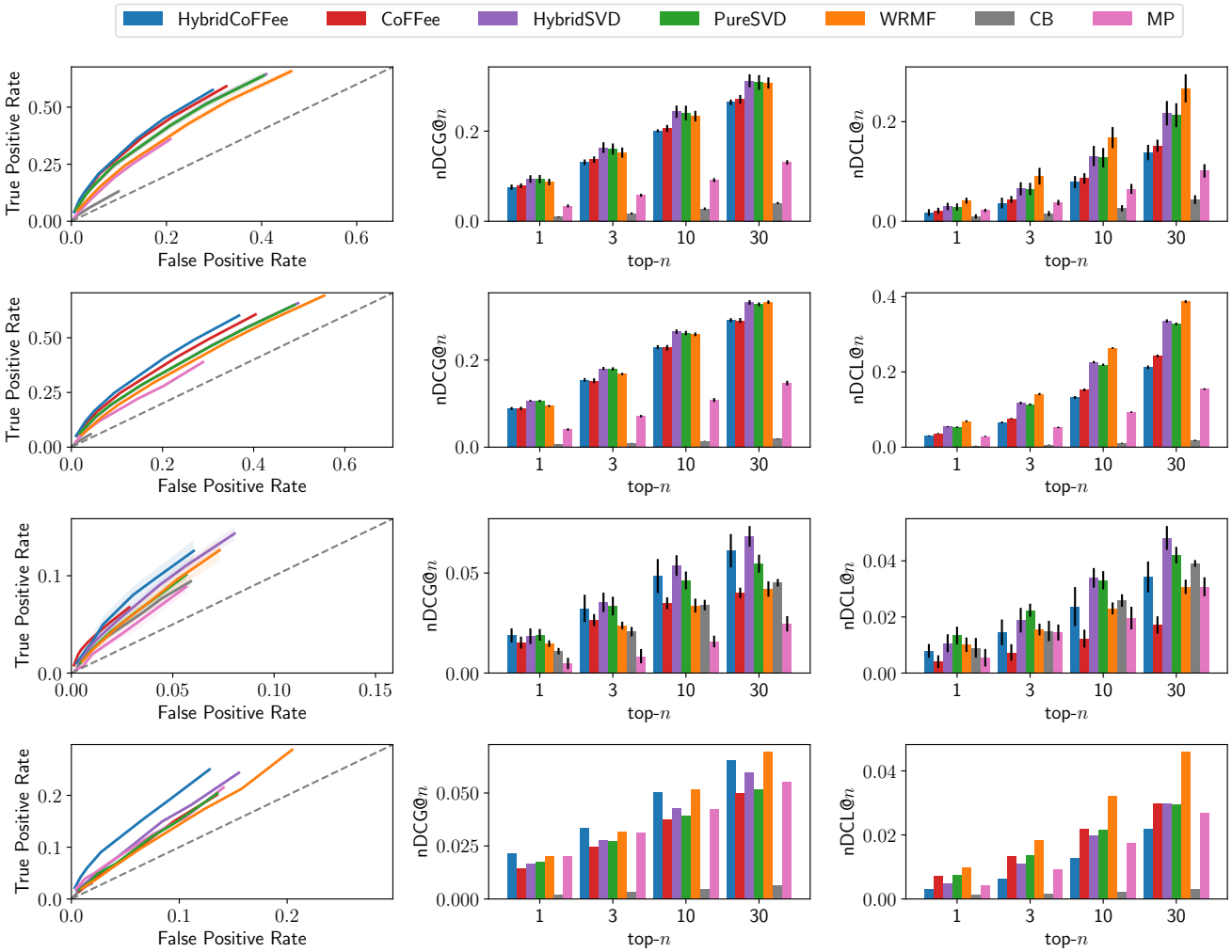

Figure 7.1: The ROC curves (1st column), nDCG@n (2nd column) and nDCL@n (3rd column). Colors of lines and bars encode different models. Rows correspond to different datasets in top-down order: ML1M, ML10M, BX and 3% fraction of ML10M. In the first 2 columns the higher the curve/bar the better; in the last column the lower the bar the better. Shaded areas in the 1st column as well as black vertical lines in the other two columns denote confidence intervals; no cross-validation is performed in the case of the sparsity test experiment, hence no intervals in the 4th row.

list, as indicated by its nDCL score. As has been argued in Sec. 7.1, this is likely to be the result of unreliable connections, created by the model, between items with very different rating values.

Likewise, in terms of nDCG score, WRMF performs comparably to other factorization models on both Movielens datasets; yet it fails in terms of the other two metrics. Its nDCL score is the worst among all models and the ratio of relevant to ir-

relevant recommendations drops below PureSVD. In the case of BX dataset the WRMF model achieves lower nDCL score comparing to the majority of other models; however, it performs poorly in terms of other metrics. In the sparsity test experiment with 3% of the ML10M dataset WRMF produces the most controversial results among all models: it amplifies the amount of both relevant and irrelevant recommendations at the same time. Irrelevant items, however, dominate, which drops the model's ROC below the curves of competing factorization methods.

As it follows from the results, HybridCoFFee *outperforms all other models in terms of the proportion of relevant recommendations to irrelevant ones.* There are two interesting results, where the model demonstrates its superiority. In the second row of the figure, which corresponds to the ML10M dataset, our model is able to decrease nDCL score below the standard CoFFee model, while keeping nDCG score at the same fairly high level. This decrease of irrelevant recommendations is immediately reflected by the ROC curve and also indicates that the model was able to make a good use of side information.

An even more remarkable result can be seen on the 4th row of the figure, which corresponds to the subsampled ML10M data with high sparsity. HybridCoFFee in that case achieves almost as high nDCG score as WRMF, whereas its nDCL score is significantly lower. Interestingly, both CoFFee and especially CB model on the same data exhibit a poor performance, which indicates that HybrdCoFFee, in turn, greatly benefits from blending collaborative data and side information together and effectively learns some non-trivial patterns.

Generally, our model exhibits the best balance between the key 3 evaluation aspects. It does not suffer from the sparsity of data as, for example, the tensor-based CoFFee model in the BX case (see the ROC curve on the third row of the Fig. 7.1). It maintains high relevance of recommendations and *generates more safe predictions*, allowing to avoid potential user disappointment.

7.6 Discussion and future work

We have presented a tensor-based approach that combines the ability to more adequately model user preferences and allows to incorporate side knowledge in order to handle data sparsity and improve the quality of recommendations. Based on the

evaluation results we show that the proposed model demonstrates the best balance between providing good recommendations and avoiding undesired user disappointment.

Note that the general formulation of our approach allows handling context information, such as time, place, mood, situation, etc., within additional dimensions, similarly to various types of feedback. This can be an interesting direction for further research, especially in the cases where context contains additional information about correlations between its different values. The key benefit of the model, in that case, is that it would allow to handle even more extreme sparsity levels induced by multidimensional representation.

Based on the remark about the curse of dimensionality problem of TD, another interesting direction for research is applying the key ideas presented in this work to more appropriate tensor formats such as TT or HT. The potential downside for such methods is the need to tune multiple rank values. Devising techniques for efficient rank selection is a challenging task and presents another vital research direction.

Part III

Software

Chapter 8

Polara: a new open-source framework for recommender systems research

One of the critical aspects of creating new recommender models is their evaluation and fair comparison. Many software libraries have been developed to date; however, none of them are concerned with the idea of feedback polarity as described in Chap. 5, and the related implementation aspects of proper model evaluation are not taken into consideration in their system design. This has led the author of this work to the development of a new framework called *Polara* – the first recommendation framework that allows a more in-depth analysis of recommender systems' performance, based on the idea of feedback polarity.

Polara, however, is not just an evaluation library and it is not limited to polarity-driven evaluation paradigm. From the very beginning, it was envisioned as a general purpose framework for quick model prototyping and comprehensive comparative analysis, featuring various evaluation regimes and testing scenarios. The framework's design and its internal structure aim to minimize the risk of unintended mistakes in routine tasks, reduce the number of potential bugs in the code and ensure consistent experimental settings. All of this allows pursuing another higher-level goal – *research reproducibility*.

The framework has also become a convenient playground for students and helped facilitate teaching in classes. For those who have just started learning about recommender systems, it makes the learning experience generally more smooth. For more advanced students it allows to focus on creative tasks. Curious minds, however, can always make a deep dive into the code and see how everything works internally.

Polara provides exceptional flexibility in both model creation and experiment setup. At the same time, it is designed to follow *recommender system for humans* paradigm, providing a high-level abstraction of general workflows with a significant focus on the usability and the ease of use.

To achieve these goals, Polara is written in Python programming language[1] – de facto, the leading platform for data science and machine learning[2]. The framework supports both Python 2 and Python 3 versions. In addition to that Polara is boosted by the Python's scientific computing ecosystem, which helps to ensure efficient operations not only in model computations but also during the evaluation phase. The latter at first glance might seem like a minor point, however in many cases evaluation takes a much longer time than actual model training, which affects the way experiments are conducted. Polara avoids running experiments user by user and, where it is possible, takes advantage of highly optimized vector operations and parallel execution to reduce an overall experiment time.

Another essential feature of Polara is the possibility to easily extend its default set of models with the help of external libraries and frameworks. This allows conducting more rigorous research that requires comparison with various existing techniques. Implementing all of them in Polara from scratch would be a tedious task, and it would be hard to keep up with the most recent advances. Instead, Polara defines a clear protocol for such interoperability and implements many convenience methods that make this process transparent and straightforward.

Table 8.1 provides a brief comparison of Polara with some popular frameworks. Besides some basic characteristics, we assess additional aspects related to functionality and usability. For example, *Customizable evaluation* column indicates whether the framework supports and allows to chose from several evaluation scenarios, which includes various data splitting protocols, data sampling strategies and flexible configuration of experimental settings. The *Warm start regime* denotes the support of a new user/new item scenario, which includes appropriate data preprocessing and/or explicit implementation of folding-in for the provided models. The names of other columns are self-descriptive.

[1] https://www.python.org/

[2] https://www.kdnuggets.com/2017/09/python-vs-r-data-science-machine-learning.html

Table 8.1: Comparison with popular recommendation frameworks.

Framework	Languages	Polarity-based evaluation	Tensor models	API for custom models	Customizable evaluation	Warm start regime	Cold start regime	Side information support	Rich set of algorithms	Regularly updated**	License
Polara [125]	Python	✓	✓	✓	✓	✓	✓	✓		✓	MIT
Mrec [106]	Python							✓			BSD
Surpise [164]	Python			✓	✓			✓		✓	BSD-3 Clause
MyMediaLite* [107]	C# / Java			✓							GNU GPL
Turi / GraphLab* [175]	C++ / Python							✓		✓	Apache 2.0
Implicit* [83]	Python					✓				✓	MIT
RankSys [129]	Java			✓				✓			MPL 2.0
LensKit [97]	Java			✓	✓	✓	✓	✓			LGPL v2.1
LibRec [100]	Java		✓	✓	✓	✓	✓	✓	✓	✓	GNU GPL
RecommenderLab [131]	R			✓		✓				✓	GNU GPL v.2

* Supported as external models in Polara.

** Last checked: September 2018.

8.1 Core components

The framework has a modular structure and is built on top of three key components, *Recommender Data*, *Recommender Model* and *Evaluation*, which consist of basic classes and standalone methods. The components are designed to support a general workflow and take care of many technical aspects, related to the stable and reliable functioning of the framework as a whole. There is also much flexibility included in these components, allowing for a high degree of customization.

The general workflow is based on the following paradigm. An instance of *Recommender Data*, holding actual user-item interactions, provides a single entry point for all *Recommender Model* instances. Recommender Data instance has a mutable state, i.e., a specific configuration, corresponding to the desired experimental setup. In turn, Recommender Model instances, i.e., actual algorithmic implementations, take the data model instance as an input argument and depend on its state. In that sense, all depen-

dent recommender instances are *subscribers* that are immediately notified on the data state changes and take appropriate actions on their side.

For example, changing train-test splitting configuration in a data model instance will lead to recomputation of a dependent recommender model instance at the very next attempt of using it (e.g., when calling for recommendations or trying to evaluate the model's performance). Alternatively, changing the number of holdout items in the test data will leave the recommender model intact; however, will flush previously calculated recommendations and will ensure that evaluation scores are refreshed at next calculation. Worth noting, the subscriber interface is exposed to a user, and it is possible to define custom actions that are executed in response to certain state changes. This mechanism can be especially useful in non-standard user-defined experiments with specific evaluation pipelines.

Overall, an interplay of the described components allows to freely experiment with various evaluation settings and be sure that all changes in experimentation setup are taken into account by recommender models without any additional actions needed from the user side. This also minimizes the amount of code, required to conduct experiments. Below are the key implementation details of each component, also demonstrating the ease of use of the framework.

8.2 Recommender Data

Recommender Data component is the central part of the framework, implemented as a standalone class with pre-defined properties and methods. It provides a rich interface with a number of tuning parameters that opens up a great level of flexibility in experiment design and ensures consistent data state across all compared models.

The component is designed in a *data-agnostic* way. As an input it takes a history of transactions in the form of a Pandas[3] dataframe, which is internally transformed into a standardized representation, allowing for efficient data manipulation and quick conversion between internal and external representations. It only requires to define, which columns of the dataframe correspond to users, items, and feedback data. As

[3]https://pandas.pydata.org/

an example, the code presented on the listing below allows to start working with the Movielens-1M dataset.

Listing 8.1: Declaring data model.

```
from polara import RecommenderData
from polara import get_movielens_data
data = get_movielens_data() # load data from Grouplens website
data_model = RecommenderData(data, "userid", "movieid", "rating")
```

The component also implements various methods for data preprocessing, data splitting and data indexing. All configurable parameters for the data manipulation and their current values can be listed with the help of `data_model.get_configuration()` call. These parameters include `test_fold` to control the fold selection in the CV experiment, `test_ratio` to define the fraction of users for test, `holdout_size` to control the number of held out items, `warm_start` to exclude test users from the training, and some other parameters that control data randomization and sampling mechanisms.

One can easily achieve almost any data configuration by assigning the appropriate values to the aforementioned parameters. For example, standard 5-fold CV experiment with 20% of users marked for test and a single top-rated holdout item per each test user can be implemented with the following configuration setup: `data_model.test_ratio = 0.2`, `data_model.holdout_size = 1`. Alternatively, in order to hold out 5% of all consumed items from *every* available user, one needs to assign `data_model.test_ratio = 0`, `data_model.holdout_size = 0.05` and `data_model.warm_start = False`[4].

Configuration parameters are wrapped with "lazy update" routines in order to prevent early triggering of subscriber notification calls and avoid multiple execution of the same commands. Configuration is applied only after calling the `data_model.prepare()` method. An attempt to read an altered configuration before calling this method will result in a warning message, informing the user that some of the changes are not yet effective.

The system of state change notifications is based on a variant of the *Observer design pattern* and uses callback functionality for communication. Several default events trigger notifications. These events correspond to modification of parameters related to either training or test data. Notification processing for the default events is imple-

[4]More examples can be found at https://github.com/Evfro/polara/tree/master/examples

mented on the recommender model's side and does not require any user involvement. It is also possible to register custom events with notifications, handled by custom user routines if the standard functionality is not sufficient for the user.

8.3 Recommender Model

Recommender Model component provides a generic interface for creating new models ready for recommendations' generation and evaluation. As with the previous component, it holds some common properties and methods that are designed to support a unified workflow independently of specific implementation details. There are several standard recommender models already implemented for user convenience, including the models based on matrix and tensor factorization. The default models are subclassed directly from the abstract base class called `RecommenderModel`. New models can be subclassed either from the base class or from already defined models to inherit some of their unique properties and extend upon them.

When creating custom models, there are two primary methods that should be implemented prior to the usage: `build` and `get_recommendations`. The former computes a recommendation model, and the latter takes its result to generate recommendations for the test users. All generated recommendations are stored as an array within the model and can be used to evaluate the model's performance or asses its behaviour for further fine-tuning.

There are several control parameters shared by all models. Among them: `filter_seen` attribute defines whether the previously consumed items are allowed to be recommended again; `topk` attribute determines the number of recommendations generated by the model; `feedack_threshold` attribute defines whether only the feedback above a certain threshold value (e.g., only ratings above 4) should be used for computing the model. These parameters are all automatically initialized with some default values when a model is created (so that users do not have to set them every time manually) and can be redefined later. The default values can be found in the `polara.recommender.defaults` module. Listing 8.2 below demonstrates an example of creating a simple SVD-based model with Polara.

Listing 8.2: Define a simple SVD-based model.

```
1 from polara import RecommenderModel
```

```
from scipy.sparse.linalg import svds # for sparse matrices

class SimpleSVD(RecommenderModel):
    def __init__(self, data_model):
        super(SimpleSVD, self).__init__(data_model)
        self.rank = 40 # set the rank of SVD
        self.method = "SVD" # label that will be used in logging

    def build(self):
        # get sparse matrix of ratings
        train_matrix = self.get_training_matrix(dtype="f8")
        # find leading left singular vectors with truncated SVD
        _, _, items_factors = svds(train_matrix, k=self.rank,
                                   return_singular_vectors="vh")
        # remember the result
        self.items_factors = items_factors

    def get_recommendations(self):
        # gather test data
        test_data, test_shape, _ = self._get_test_data()
        # construct sparse rating matrix for all test users
        test_matrix, test_idx = self.get_test_matrix(test_data,
                                                     test_shape)
        # compute predicted scores for all test users at once
        v = self.items_factors
        svd_scores = (test_matrix.dot(v.T)).dot(v)
        if self.filter_seen:
            # prevent seen items from appearing in recommendations
            self.downvote_seen_items(svd_scores, test_idx)
        # compute recommendations
        top_recs = self.get_topk_elements(svd_scores)
        return top_recs
```

The newly created model can now be used in a general workflow. It only takes a few lines of code and the commands for that are self-explanatory, as illustrated below.

Listing 8.3: Create and evaluate the model.

```
svd = SimpleSVD(data_model)
svd.build()
svd.evaluate("relevance")
```

```
4 # output will be something like
5 # Relevance(precision=0.0994, recall=0.3314...
```

We note that defining the `build` method is entirely on the user's responsibility and depends only on the choice of a particular algorithmic implementation. In turn, the `get_recommendations` method provides two options. The first option is to manually implement all of its internal logic similarly to the example in Listing 8.2. In that case, the user is responsible for making the code efficient in terms of both computational resources and available memory. This can be a reasonable option when data is small, and the framework is used for learning purposes.

However, in order to deal with real data, special care must be taken on the process of recommendations generation. If the number of items as well as the number of test users is huge, intermediate calculation results may consume all available memory (e.g., line 27 of Listing 8.2, where a complete dense matrix is created). Moreover, as the memory I/O is generally slower than CPU operations, even if there is enough memory it is more efficient to limit its consumption by the model and expose memory resources in pieces of a fixed size.

In order to achieve that the default implementation of the `get_recommendations` method in the base class splits the test data into chunks. Every chunk will include a number of unique test users, typically more than one. This relies on the assumption that computing recommendations for a group of users at once is much more efficient than looping over every user individually. This is the case in a number of scenarios (e.g., standard scenario of recommending to the known users) and for a wide range of algorithms, including SVD, MF, TF, etc., as it may take advantage of BLAS operations.

In order to operate over the chunks of data, one only needs to declare a `slice_recommendations` method, which is by default the key part of the `get_recommendations` method. The code in Listing 8.4 indicates the necessary changes in the new model creation to activate this functionality.

Listing 8.4: More efficient variant of defining a model.

```
1 class SimpleSVD(RecommenderModel):
2     def __init__(self, data_model):
3         # same as before
4 
5     def build(self):
6         # same as before
```

```
7
8       def slice_recommendations(self, test_data, test_shape,
9                                 start, stop, test_users=None):
10          test_slice = (start, stop) # selecting users from a range
11          test_matrix, slice_data = self.get_test_matrix(test_data,
12                                                         test_shape,
13                                                         test_slice)
14          v = self.items_factors
15          scores = (test_matrix.dot(v)).dot(v.T)
16          return scores, slice_data
```

The `slice_recommendations` method here operates on a group of users, selected by an index range. The predicted scores are computed only for these users and then are returned to the `get_recommendations` method to generate final recommendations. The method also returns index data to allow filtering out previously seen items from recommendations. Note, that there's no need to define `get_recommendations` anymore and the code becomes slightly simpler.

The size of chunks (i.e., the number of test users in it) is controlled by a predefined memory limit, which can be set via the `MEMORY_HARD_LIMIT` attribute from the `polara.recommender.utils` module. Its optimal value depends on the hardware capabilities and should be determined empirically. It may range from one gigabyte to several dozens of gigabytes. Setting it to lower values will lead to a computational overhead with many small iterations, while too high values (if there's enough available memory) are unlikely to improve performance due to I/O bounds.

The I/O bound, however, can be alleviated with parallel execution. When data is large, I/O operations like reading the data of a group of test users may take more time than actual computations. Such operations typically lock Python's Global Interpreter Lock. In order to mitigate that limitation, the `slice_recommendations` method can be executed in parallel threads. This behavior is controlled by the `max_test_workers` parameter of a recommender model. Setting it to a non-zero value defines the number of parallel threads. The maximum amount of memory consumed by a model during the recommendations generation can be estimated as `MEMORY_HARD_LIMIT * max_test_workers` gigabytes.

Note that in the examples above only one model is created. However, as has been previously mentioned, several recommender models can share the same `data_model` in order to conduct bulk experiments with fair model comparison.

Polara can be easily extended with the help of external libraries and frameworks. It uses the concept of *wrapper* – an interface between internal methods and external sources. The general process of creating new wrappers is no different from creating new models within the framework and requires minimum efforts. By default, Polara already implements several wrappers for the well-known software tools, which extends the list of supported algorithms. This includes *MyMediaLite* [107], *GraphLab Create* [175] and *implicit* [83].

8.4 Evaluation

Unlike the previous components, the evaluation component is not a particular class but rather is a set of convenience methods, designed to support various evaluation scenarios in a unified way. The major focus of evaluation is shifted towards the relevance of recommendations and the quality of recommendation ranking. There are several standard evaluation metrics supported by this component, namely Precision, Recall, HR, MRR, nDCG, nDCL and a number of others.

Two key features distinguish this component from evaluation components in other recommendation libraries and frameworks. The first one is a native control over the false positive rate estimation. As described in Sec. 5.3.1, recommender models may recommend items that have no user feedback (this happens very often, in fact). Treating them as false positives in some cases leads to an undesired fp rate overestimation and spoils the precision-recall curve.

Polara allows users to assign more appropriate weighting in this case via the `not_rated_penalty` argument of the `model.evaluate()` method. Setting its value to 1 will force the evaluation process to count recommendations with unknown user feedback as false positives while setting it to 0 will filter out such recommendations from the final score calculation. Values between 0 and 1 will lead to a "smooth" fp rate estimation.

The second feature is a native support for the positivity threshold, also described in Sec. 5.3.1. As has been demonstrated in the results of Chap. 5, taking into account

the performance of models with respect to both positive and negative aspects of recommendations plays a crucial role in an understanding of the overall quality of recommendations. Hence, every recommender model is provided with the `switch_positive` trigger, which allows defining, what values of feedback should be treated as positive or negative examples when evaluating recommendation quality. This trigger not only affects how metrics are computed but also allows to calculate the nDCL score, introduced in Sec. 5.3.2. It also defines which recommendations are counted as false positive.

The technical implementation of the component relies on a sparse data representation and bulk computations without loops. This not only improves computational efficiency but also allows to conduct a more in-depth analysis of model performance, going beyond aggregated evaluation and in some cases helping to create a better picture of model behavior. One particular method worth mentioning in this regard is `assemble_scoring_matrices` from the `polara.recommender.evaluation` module. It takes generated recommendations and holdout data as an input and returns various indicators of correct and incorrect recommendations in the form of sparse matrices.

8.5 Supported scenarios and setups

As a multi-purpose evaluation framework, Polara provides the necessary instruments and controls for various setups that cover all major evaluation scenarios. There are three main experiment setups, supported natively by Polara. The *standard evaluation* scenario allows test users to be a part of the training data and only the items from holdout set remain unknown until the evaluation phase. In the *warm start* scenario the test users are also hidden from the training phase. During the evaluation phase, their known preferences are used to generate recommendations, which are then evaluated against the holdout items. Finally, the *cold start* scenario is represented by a separate `polara.recommender.coldstart` module, which currently provides item cold start functionality. The module additionally provides a few methods to manipulate content information to support cold start regime.

Polara supports both implicit and explicit feedback, independently of whether it is represented by rating values, binary data, frequency counts or other data formats.

Note that categorical feedback naturally fits the tensor-based representation and can also be handled within Polara.

In addition to standard data splitting methods, Polara also supports custom fields that can be used to order elements and split data. The simplest example is the timestamp data. Assuming there is an additional column named "timestamp" in the original pandas dataframe, the following modification of the Recommender Data constructor allows to take this information into account:

```
RecommenderData(data, "userid", "itemid", custom_order="timestamp"),
```

where for illustration purposes we also omit the `feedback` field to demonstrate how to handle purely implicit positive-only data.

An important part of the general evaluation framework is the ability to set custom test data, provided externally (e.g. in some online recommender system challenge). For example, if one is provided with some external holdout data, which is not a part of the training data, however contains only known users, the following setup allows to seamlessly work the data:

Listing 8.5: Preparing data model for experiments with custom holdout.

```
data_model = RecommenderData(data, "userid", "itemid", "feedback")
data_model.prepare_training_only() # do not attempt to split data
data_model.set_test_data(holdout=external_holdout,
                         warm_start=False)
```

It should be noted that by default Polara will reindex `external_holdout` data to conform with the internal data representation. This behavior can be disabled by providing `reindex = False` argument into the `set_test_data` method.

Fine-tuning of many recommendation models is not as simple as the tuning of SVD and often requires an extensive hyper-parameter search. Current implementation of Polara provides basic functionality for the random grid search, which can be accessed via the `random_grid` method from the `polara.evaluation.pipelines` module. This functionality will be extended in future versions and include customizable all-in-one pipelines.

As a final example of the framework functionality, the listing below demonstrates how to conduct a top-n recommendation experiment for several models in bulk with a few lines of code in the current version of the framework:

Listing 8.6: Example of cross-validation experiment for evaluating several models

```
from polara import PopularityModel
from polara import RandomModel
from polara.evaluation import evaluation_engine as ee

svd = SVDModel(data_model)
popular = PopularityModel(data_model)
random = RandomModel(data_model)

models = [svd, popular, random]
metrics = ["ranking", "relevance"]
topk_values = [1, 5, 10, 20, 50] # number of recommendations

topk_result = {}
for fold in [1, 2, 3, 4, 5] :
    data_model.test_fold = fold
    topk_result[fold] = ee.topk_test(models, topk_values, metrics)
```

This will store the result of all models' evaluation for all 5 folds in the `topk_result` variable, which can be further used to perform comparative analysis and report on findings.

8.6 Summary

In this chapter, we have described the key design aspects and demonstrated the main functionality of the Polara framework. It takes care of the most of the data processing and data handling hassles, providing a thin, abstract layer for the user with a rich set of controls. The framework also provides a number of convenient and flexible software tools for quick prototyping of recommender models and performing a comprehensive evaluation. Apart from the boilerplate functionality, Polara also supports several external frameworks and libraries, allowing to incorporate their models into the general workflow. Internally, the framework uses various tweaks and controls in order to perform operations efficiently and wisely consume system resources. The framework is suitable for both beginners and advanced users; it can be used in classes for teaching or as a part of a daily research.

Final conclusion

In this work, we have considered various aspects of the limited preference information problem. This includes both cold start and warm start regimes, as well as the general problem of the insufficient amount of collaborative information, which often raises due to low user activity or overwhelmingly large collection of items. The main contribution of this work consists of the following parts:

- A new method for a proper modelling of user feedback is proposed. It allows to better handle both positive and negative user feedback and improve user experience during the rating elicitation phase or in a general warm start scenario. The method is based on the Tucker Decomposition and can be viewed as an expansion of the PureSVD approach to higher order cases.

- The second proposed method uses a generalized formulation of SVD in order to add the ability to use side information along with collaborative data. This allows to handle cases of extreme data sparsity and maintain high quality of recommendations. The method is also suitable for cold start regime.

- The third proposed method combines the previous two methods into a unified approach. We provide efficient optimization technique, which takes the specific structure of the problem into account. Remarkably, the method demonstrates all the advantages of its predecessors and at the same time does not suffer from their major shortcomings.

- All three methods use SVD as an atomic operation during the optimization process and preserve the orthogonality of columns in factor matrices. This allows to maintain high scalability and makes all methods especially suitable for online settings due to simplified folding-in computation.

- The methods also have minimal requirements for optimal hyper-parameter search. This is achieved with a simplified rank tuning, which can be performed by rank truncation in the SVD case and tensor rounding in the higher order case.

- A new open-source recommendation framework written in Python is developed. The framework proved to be useful for quick model prototyping, comprehensive quality evaluation and also research reproducibility.

Despite considerable attention given to the rating data, the proposed unified model is potentially applicable to other types of feedback as well, including different response to the system, different user actions, emojis, multiple criteria ratings, etc. Due to a general formulation of its underlying principles, the model is also suitable for context-aware or multi-aspect settings. Based on side information, the model helps to reasonably restore missing connections between various aspects or entities and impose additional constraints on them. We believe the model can be applied in many domains, going beyond entertainment systems.

Despite the encouraging results, there is a general issue related to solving problems with multiple types of context and feedback values. When the number of dimensions becomes much higher than 3, application of TD-based methods becomes infeasible due to the explosion of storage requirements. A possible cure for this problem is to use TT/HT formats for tensor decomposition. Incorporating the ideas, developed in this work, into a more appropriate tensor format for higher dimensional problems remains a promising direction for further investigations.

Bibliography

[1] Hervé Abdi. "Singular value decomposition (SVD) and generalized singular value decomposition". In: *Encyclopedia of measurement and statistics. Thousand Oaks (CA): Sage* (2007), pp. 907–12.

[2] Gediminas Adomavicius and Alexander Tuzhilin. "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions". In: *IEEE transactions on knowledge and data engineering* 17.6 (2005), pp. 734–749.

[3] Gediminas Adomavicius et al. "Context-Aware Recommender Systems." In: *AI Magazine* 32.3 (2011).

[4] Gediminas Adomavicius et al. "Incorporating contextual information in recommender systems using a multidimensional approach". In: *ACM Transactions on Information Systems (TOIS)* 23.1 (2005), pp. 103–145.

[5] Deepak Agarwal and Bee-Chung Chen. "Regression-based latent factor models". In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2009, pp. 19–28.

[6] James Allan et al. "Frontiers, challenges, and opportunities for information retrieval: Report from SWIRL 2012 the second strategic workshop on information retrieval in Lorne". In: *ACM SIGIR Forum*. Vol. 46. 1. ACM. 2012, pp. 2–32.

[7] Xavier Amatriain and Deepak Agarwal. "Tutorial: Lessons Learned from Building Real-life Recommender Systems". In: *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM. 2016, pp. 433–433.

[8] Xavier Amatriain, Josep M Pujol, and Nuria Oliver. "I like it... i like it not: Evaluating user ratings noise in recommender systems". In: *International Confer-*

ence on User Modeling, Adaptation, and Personalization. Springer. 2009, pp. 247–258.

[9] Claus A Andersson and Rasmus Bro. "Improving the speed of multi-way algorithms:: Part I. Tucker3". In: *Chemometrics and intelligent laboratory systems* 42.1-2 (1998), pp. 93–103.

[10] Yoram Bachrach et al. "Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces". In: *Proceedings of the 8th ACM Conference on Recommender systems*. ACM. 2014, pp. 257–264.

[11] Iman Barjasteh et al. "Cold-start item and user recommendation with decoupled completion and transduction". In: *Proceedings of the 9th ACM Conference on Recommender Systems*. ACM. 2015, pp. 91–98.

[12] Immanuel Bayer et al. "A generic coordinate descent framework for learning from implicit feedback". In: *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee. 2017, pp. 1341–1350.

[13] Mary Bazire and Patrick Brézillon. "Understanding context before using it". In: *International and Interdisciplinary Conference on Modeling and Using Context*. Springer. 2005, pp. 29–40.

[14] Robert M Bell and Yehuda Koren. "Scalable collaborative filtering with jointly derived neighborhood interpolation weights". In: *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*. IEEE. 2007, pp. 43–52.

[15] Robert Bell, Yehuda Koren, and Chris Volinsky. "Modeling relationships at multiple scales to improve accuracy of large recommender systems". In: *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2007, pp. 95–104.

[16] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization". In: *Journal of Machine Learning Research* 13.Feb (2012), pp. 281–305.

[17] Michael W Berry, Susan T Dumais, and Gavin W O'Brien. "Using linear algebra for intelligent information retrieval". In: *SIAM review* 37.4 (1995), pp. 573–595.

[18] Dimitri P Bertsekas. *Nonlinear programming*. Athena scientific Belmont, 1999.

[19] Daniel Billsus and Michael J Pazzani. “Learning Collaborative Information Filters.” In: *Icml*. Vol. 98. 1998, pp. 46–54.

[20] Mathieu Blondel et al. “Higher-order factorization machines”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 3351–3359.

[21] Jesús Bobadilla et al. “Recommender systems survey”. In: *Knowledge-based systems* 46 (2013), pp. 109–132.

[22] Léon Bottou. “Stochastic gradient descent tricks”. In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 421–436.

[23] Matthew Brand. “Incremental singular value decomposition of uncertain data with missing values”. In: *European Conference on Computer Vision*. Springer. 2002, pp. 707–720.

[24] John S Breese, David Heckerman, and Carl Kadie. “Empirical analysis of predictive algorithms for collaborative filtering”. In: *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc. 1998, pp. 43–52.

[25] Peter Brusilovsky. “Social information access: the other side of the social web”. In: *International Conference on Current Trends in Theory and Practice of Computer Science*. Springer. 2008, pp. 5–22.

[26] Robin Burke. “Hybrid web recommender systems”. In: *The adaptive web*. Springer, 2007, pp. 377–408.

[27] Dennis L Chao, Justin Balthrop, and Stephanie Forrest. “Adaptive radio: achieving consensus using negative preferences”. In: *Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work*. ACM. 2005, pp. 120–123.

[28] Olivier Chapelle and Mingrui Wu. “Gradient descent optimization of smoothed information retrieval metrics”. In: *Information retrieval* 13.3 (2010), pp. 216–235.

[29] Tianqi Chen et al. “Combining factorization model and additive forest for collaborative followee recommendation”. In: *KDD CUP* (2012).

[30] Tianqi Chen et al. *Feature-based matrix factorization*. 2011. arXiv: 1109.2271. URL: https://arxiv.org/abs/1109.2271.

[31] Yifan Chen, Xiang Zhao, and Maarten de Rijke. "Top-N recommendation with high-dimensional side information via locality preserving projection". In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM. 2017, pp. 985–988.

[32] Yun Chi and Shenghuo Zhu. "FacetCube: a framework of incorporating prior knowledge into non-negative tensor factorization". In: *Proceedings of the 19th ACM international conference on Information and knowledge management*. ACM. 2010, pp. 569–578.

[33] Yun Chi et al. "Probabilistic polyadic factorization and its application to personalized recommendation". In: *Proceedings of the 17th ACM conference on Information and knowledge management*. ACM. 2008, pp. 941–950.

[34] Flavio Chierichetti et al. "Finding the jaccard median". In: *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*. SIAM. 2010, pp. 293–311.

[35] Andrzej Cichocki and Anh-Huy Phan. "Fast local algorithms for large scale nonnegative matrix and tensor factorizations". In: *IEICE transactions on fundamentals of electronics, communications and computer sciences* 92.3 (2009), pp. 708–721.

[36] Andrzej Cichocki et al. *Nonnegative matrix and tensor factorizations: applications to exploratory multi-way data analysis and blind source separation*. John Wiley & Sons, 2009.

[37] Pierre Comon. "Tensors: a brief introduction". In: *IEEE Signal Processing Magazine* 31.3 (2014), pp. 44–53.

[38] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. "Performance of recommender algorithms on top-n recommendation tasks". In: *Proceedings of the fourth ACM conference on Recommender systems*. New York, NY, USA: ACM, 2010, pp. 39–46.

[39] Ariyam Das et al. “Collaborative Filtering As a Case-Study for Model Parallelism on Bulk Synchronous Systems”. In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. CIKM ’17. New York, NY, USA: ACM, 2017, pp. 969–977.

[40] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. “A Multilinear Singular Value Decomposition”. In: *SIAM J. Matrix Anal. Appl.* 21.4 (Jan. 2000), pp. 1253–1278.

[41] Vin De Silva and Lek-Heng Lim. “Tensor rank and the ill-posedness of the best low-rank approximation problem”. In: *SIAM Journal on Matrix Analysis and Applications* 30.3 (2008), pp. 1084–1127.

[42] Scott Deerwester et al. “Indexing by latent semantic analysis”. In: *Journal of the American society for information science* 41.6 (1990), p. 391.

[43] Mukund Deshpande and George Karypis. “Item-based top-n recommendation algorithms”. In: *ACM Transactions on Information Systems (TOIS)* 22.1 (2004), pp. 143–177.

[44] Christian Desrosiers and George Karypis. “A comprehensive survey of neighborhood-based recommendation methods”. In: *Recommender systems handbook*. Springer, 2011, pp. 107–144.

[45] Stephan Doerfel, Robert Jäschke, and Gerd Stumme. “The role of cores in recommender benchmarking for social bookmarking systems”. In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 7.3 (2016), p. 40.

[46] Paul Dourish. “What we talk about when we talk about context”. In: *Personal and ubiquitous computing* 8.1 (2004), pp. 19–30.

[47] John Duchi, Elad Hazan, and Yoram Singer. “Adaptive subgradient methods for online learning and stochastic optimization”. In: *Journal of Machine Learning Research* 12.Jul (2011), pp. 2121–2159.

[48] Carl Eckart and Gale Young. “The approximation of one matrix by another of lower rank”. In: *Psychometrika* 1.3 (1936), pp. 211–218.

[49] Michael D Ekstrand, John T Riedl, Joseph A Konstan, et al. “Collaborative filtering recommender systems”. In: *Foundations and Trends® in Human–Computer Interaction* 4.2 (2011), pp. 81–173.

[50] Mehdi Elahi, Valdemaras Repsys, and Francesco Ricci. "Rating elicitation strategies for collaborative filtering". In: *International Conference on Electronic Commerce and Web Technologies*. Springer. 2011, pp. 160–171.

[51] Mehdi Elahi, Francesco Ricci, and Neil Rubens. "A survey of active learning in collaborative filtering recommender systems". In: *Computer Science Review* 20 (2016), pp. 29–50.

[52] Yi Fang and Luo Si. "Matrix co-factorization for recommendation with rich side information and implicit feedback". In: *Proceedings of the 2nd International Workshop on Information Heterogeneity and Fusion in Recommender Systems*. ACM. 2011, pp. 65–69.

[53] Alexander Fonarev et al. "Efficient rectangular maximal-volume algorithm for rating elicitation in collaborative filtering". In: *Data Mining (ICDM), 2016 IEEE 16th International Conference on*. IEEE. 2016, pp. 141–150.

[54] Evgeny Frolov and Ivan Oseledets. "Tensor methods and recommender systems". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 7.3 (2017).

[55] Xiao Fu et al. "Joint tensor factorization and outlying slab suppression with applications". In: *IEEE Transactions on Signal Processing* 63.23 (2015), pp. 6315–6328.

[56] George W Furnas et al. "Information retrieval using a singular value decomposition model of latentsemantic structure". In: *Proc. 11th Annu. Int. ACM SIGIR Conf. Res. Dev. Inf. Retr.* ACM. 1988, pp. 465–480.

[57] 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*. ACM. 2010, pp. 14–19.

[58] Zeno Gantner et al. "Learning attribute-to-feature mappings for cold-start recommendations". In: *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE. 2010, pp. 176–185.

[59] Zeno Gantner et al. "MyMediaLite: a free recommender system library". In: *Proceedings of the fifth ACM conference on Recommender systems*. ACM. 2011, pp. 305–308.

[60] Mark Gates et al. “Accelerating collaborative filtering using concepts from high performance computing”. In: *Big Data (Big Data), 2015 IEEE International Conference on*. IEEE. 2015, pp. 667–676.

[61] Hancheng Ge, James Caverlee, and Haokai Lu. “Taper: A contextual tensor-based approach for personalized expert recommendation”. In: *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM. 2016, pp. 261–268.

[62] Rainer Gemulla et al. “Large-scale matrix factorization with distributed stochastic gradient descent”. In: *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2011, pp. 69–77.

[63] Alex Gittens et al. “Matrix factorizations at scale: A comparison of scientific data analytics in Spark and C+ MPI using three case studies”. In: *Big Data (Big Data), 2016 IEEE International Conference on*. IEEE. 2016, pp. 204–213.

[64] Ken Goldberg et al. “Eigentaste: A constant time collaborative filtering algorithm”. In: *Information Retrieval* 4.2 (2001), pp. 133–151.

[65] Gene H Golub and Christian Reinsch. “Singular value decomposition and least squares solutions”. In: *Numer. Math.* 14.5 (1970), pp. 403–420.

[66] Gene H Golub and Charles F Van Loan. *Matrix computations*. 4th. The Johns Hopkins University Press, 2012.

[67] Lars Grasedyck. “Hierarchical singular value decomposition of tensors”. In: *SIAM J. Matrix Anal. Appl.* 31.4 (2010), pp. 2029–2054.

[68] Lars Grasedyck, Daniel Kressner, and Christine Tobler. “A literature survey of low-rank tensor approximation techniques”. In: *GAMM-Mitteilungen* 36.1 (2013), pp. 53–78.

[69] Asela Gunawardana and Christopher Meek. “A unified approach to building hybrid recommender systems”. In: *Proceedings of the third ACM conference on Recommender systems*. ACM. 2009, pp. 117–124.

[70] Wolfgang Hackbusch. *Tensor spaces and numerical tensor calculus*. Vol. 42. Springer Science & Business Media, 2012.

[71] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. “Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions”. In: *SIAM review* 53.2 (2011), pp. 217–288.

[72] Trevor Hastie et al. “Matrix completion and low-rank SVD via fast alternating least squares.” In: *Journal of Machine Learning Research* 16 (2015), pp. 3367–3402.

[73] Trevor Hastie et al. “The elements of statistical learning: data mining, inference and prediction”. In: *Math. Intell.* 27.2 (2005), pp. 83–85.

[74] Jonathan L Herlocker et al. “Evaluating collaborative filtering recommender systems”. In: *ACM Transactions on Information Systems (TOIS)* 22.1 (2004), pp. 5–53.

[75] Balázs Hidasi. “Factorization models for context-aware recommendations”. In: *Infocommun J VI (4)* (2014), pp. 27–34.

[76] Balázs Hidasi and Domonkos Tikk. *Context-aware recommendations from implicit data via scalable tensor factorization*. 2013. arXiv: 1309.7611. URL: https://arxiv.org/abs/1309.7611.

[77] Balázs Hidasi and Domonkos Tikk. “Fast ALS-based tensor factorization for context-aware recommendation from implicit feedback”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2012, pp. 67–82.

[78] Balázs Hidasi and Domonkos Tikk. “General factorization framework for context-aware recommendations”. In: *Data Mining and Knowledge Discovery* 30.2 (2016), pp. 342–371.

[79] Balázs Hidasi and Domonkos Tikk. “Speeding up ALS learning via approximate methods for context-aware recommendations”. In: *Knowledge and Information Systems* 47.1 (2016), pp. 131–155.

[80] Liangjie Hong, Aziz S Doumith, and Brian D Davison. “Co-factorization machines: modeling user interests and predicting individual decisions in twitter”. In: *Proceedings of the sixth ACM international conference on Web search and data mining*. ACM. 2013, pp. 557–566.

[81] Cheng-Kang Hsieh et al. “Collaborative metric learning”. In: *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee. 2017, pp. 193–201.

[82] Yifan Hu, Yehuda Koren, and Chris Volinsky. “Collaborative filtering for implicit feedback datasets”. In: *2008 Eighth IEEE International Conference on Data Mining*. IEEE. 2008, pp. 263–272.

[83] *implicit: Fast Python Collaborative Filtering for Implicit Feedback Datasets*. URL: `https://github.com/benfred/implicit` (visited on 09/12/2018).

[84] Yuchin Juan et al. “Field-aware factorization machines for CTR prediction”. In: *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM. 2016, pp. 43–50.

[85] Alexandros Karatzoglou et al. “Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering”. In: *Proceedings of the fourth ACM conference on Recommender systems*. ACM. 2010, pp. 79–86.

[86] Rasoul Karimi et al. “Non-myopic active learning for recommender systems based on matrix factorization”. In: *Information Reuse and Integration (IRI), 2011 IEEE International Conference on*. IEEE. 2011, pp. 299–303.

[87] Elizabeth A Kensinger. “Remembering the details: Effects of emotion”. In: *Emotion review* 1.2 (2009), pp. 99–113.

[88] Dohyun Kim and Bong-Jin Yum. “Collaborative filtering based on iterative principal component analysis”. In: *Expert Systems with Applications* 28.4 (2005), pp. 823–830.

[89] Bart P Knijnenburg and Martijn C Willemsen. “Evaluating recommender systems with user experiments”. In: *Recommender Systems Handbook*. Springer, 2015, pp. 309–352.

[90] Tamara G Kolda and Brett W Bader. “Tensor decompositions and applications”. In: *SIAM review* 51.3 (2009), pp. 455–500.

[91] Tamara Kolda and Brett Bader. “The TOPHITS model for higher-order web link analysis”. In: *Workshop on link analysis, counterterrorism and security*. Vol. 7. 2006, pp. 26–29.

[92] Joseph A Konstan and John Riedl. “Recommender systems: from algorithms to user experience”. In: *User modeling and user-adapted interaction* 22.1-2 (2012), pp. 101–123.

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

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

[95] Yehuda Koren and Joe Sill. “OrdRec: an ordinal model for predicting personalized item rating distributions”. In: *Proceedings of the fifth ACM conference on Recommender systems*. ACM. 2011, pp. 117–124.

[96] Jongwuk Lee et al. “Improving the accuracy of top-N recommendation using a preference model”. In: *Information Sciences* 348 (2016), pp. 290–304.

[97] *LenSkit. Open-Source Tools for Recommender Systems*. URL: `https://lenskit.org` (visited on 09/12/2018).

[98] Gai Li and Qiang Chen. “Exploiting explicit and implicit feedback for personalized ranking”. In: *Mathematical Problems in Engineering* 2016 (2016).

[99] Zhi-fang Liao et al. “A tripartite tensor decomposition fold-in for social tagging”. In: *Tamkang Journal of Science and Engineering* 17.4 (2014), pp. 363–370.

[100] *Librec. A Leading Java Library for Recommender Systems*. URL: `https://www.librec.net` (visited on 09/12/2018).

[101] Tie-Yan Liu et al. “Learning to rank for information retrieval”. In: *Foundations and Trends® in Information Retrieval* 3.3 (2009), pp. 225–331.

[102] Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro. “Content-based recommender systems: State of the art and trends”. In: *Recommender systems handbook*. Springer, 2011, pp. 73–105.

[103] Michael W Mahoney, Mauro Maggioni, and Petros Drineas. “Tensor-CUR decompositions for tensor-based data”. In: *SIAM Journal on Matrix Analysis and Applications* 30.3 (2008), pp. 957–987.

[104] Leandro Balby Marinho et al. “Social tagging recommender systems”. In: *Recommender systems handbook*. Springer, 2011, pp. 615–644.

[105] Andriy Mnih and Ruslan R Salakhutdinov. “Probabilistic matrix factorization”. In: *Advances in neural information processing systems*. 2008, pp. 1257–1264.

[106] *mrec recommender systems library*. URL: `https://mendeley.github.io/mrec` (visited on 09/12/2018).

[107] *MyMediaLite Recommender System Library*. URL: `http://www.mymedialite.net` (visited on 09/12/2018).

[108] Amir Hossein Nabizadeh et al. “Predicting User Preference Based on Matrix Factorization by Exploiting Music Attributes”. In: *Proceedings of the Ninth International C* Conference on Computer Science & Software Engineering*. ACM. 2016, pp. 61–66.

[109] Alexandros Nanopoulos et al. “Musicbox: Personalized music recommendation based on cubic analysis of social tags”. In: *IEEE Transactions on Audio, Speech, and Language Processing* 18.2 (2010), pp. 407–412.

[110] Carmeliza Navasca, Lieven De Lathauwer, and Stefan Kindermann. “Swamp reducing technique for tensor decomposition”. In: *Signal Processing Conference, 2008 16th European*. IEEE. 2008, pp. 1–5.

[111] Jennifer Nguyen and Mu Zhu. “Content-boosted matrix factorization techniques for recommender systems”. In: *Statistical Analysis and Data Mining* 6.4 (2013), pp. 286–301.

[112] Athanasios N Nikolakopoulos et al. “EigenRec: generalizing PureSVD for effective and efficient top-N recommendations”. In: *Knowledge and Information Systems* (2018), pp. 1–23.

[113] Xia Ning and George Karypis. “Slim: Sparse linear methods for top-n recommender systems”. In: *Data Mining (ICDM), 2011 IEEE 11th International Conference on*. IEEE. 2011, pp. 497–506.

[114] Xia Ning and George Karypis. “Sparse linear methods with side information for top-n recommendations”. In: *Proceedings of the sixth ACM conference on Recommender systems*. ACM. 2012, pp. 155–162.

[115] Alexander Novikov, Mikhail Trofimov, and Ivan Oseledets. *Exponential machines*. 2016. arXiv: 1605.03795. URL: https://arxiv.org/abs/1605.03795.

[116] Ivan V Oseledets. "Tensor-train decomposition". In: *SIAM Journal on Scientific Computing* 33.5 (2011), pp. 2295–2317.

[117] Rong Pan et al. "One-class collaborative filtering". In: *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*. IEEE. 2008, pp. 502–511.

[118] Xinghao Pan et al. "Cyclades: Conflict-free asynchronous machine learning". In: *Advances in Neural Information Processing Systems*. 2016, pp. 2568–2576.

[119] Ulrich Paquet, Blaise Thomson, and Ole Winther. "A hierarchical model for ordinal matrix factorization". In: *Statistics and Computing* 22.4 (2012), pp. 945–957.

[120] Denis Parra and Peter Brusilovsky. "Collaborative filtering for social tagging systems: an experiment with CiteULike". In: *Proceedings of the third ACM conference on Recommender systems*. ACM, 2009, pp. 237–240.

[121] Arkadiusz Paterek. "Improving regularized singular value decomposition for collaborative filtering". In: *Proceedings of KDD cup and workshop*. Vol. 2007. 2007, pp. 5–8.

[122] Dinh Q Phung, Svetha Venkatesh, et al. "Ordinal Boltzmann machines for collaborative filtering". In: *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*. AUAI Press. 2009, pp. 548–556.

[123] István Pilászy and Domonkos Tikk. "Recommending new movies: even a few ratings are more valuable than metadata". In: *Proceedings of the third ACM conference on Recommender systems*. ACM. 2009, pp. 93–100.

[124] István Pilászy, Dávid Zibriczky, and Domonkos Tikk. "Fast als-based matrix factorization for explicit and implicit feedback datasets". In: *Proceedings of the fourth ACM conference on Recommender systems*. ACM. 2010, pp. 71–78.

[125] *Polara: Recommender Systems framework*. URL: https://github.com/evfro/polara (visited on 09/12/2018).

[126] Ian Porteous, Arthur U Asuncion, and Max Welling. "Bayesian Matrix Factorization with Side Information and Dirichlet Process Mixtures." In: *AAAI*. 2010.

[127] Dimitrios Rafailidis and Petros Daras. “The TFC model: Tensor factorization and tag clustering for item recommendation in social tagging systems”. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 43.3 (2013), pp. 673–688.

[128] Dimitrios Rafailidis and Alexandros Nanopoulos. “Modeling users preference dynamics and side information in recommender systems”. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 46.6 (2016), pp. 782–792.

[129] *RankSys. Java 8 Recommender Systems framework for novelty, diversity and much more.* URL: `https://www.ranksys.org/` (visited on 09/12/2018).

[130] Benjamin Recht et al. “Hogwild: A lock-free approach to parallelizing stochastic gradient descent”. In: *Advances in neural information processing systems*. 2011, pp. 693–701.

[131] *recommenderlab: A Framework for Developing and Testing Recommendation Algorithms.* URL: `http://s2.smu.edu/IDA/recommenderlab` (visited on 09/12/2018).

[132] Steffen Rendle. “Factorization machines”. In: *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE. 2010, pp. 995–1000.

[133] Steffen Rendle and Lars Schmidt-Thieme. “Online-updating regularized kernel matrix factorization models for large-scale recommender systems”. In: *Proceedings of the 2008 ACM conference on Recommender systems*. ACM. 2008, pp. 251–258.

[134] Steffen Rendle and Lars Schmidt-Thieme. “Pairwise interaction tensor factorization for personalized tag recommendation”. In: *Proceedings of the third ACM international conference on Web search and data mining*. ACM. 2010, pp. 81–90.

[135] Steffen Rendle et al. “BPR: Bayesian personalized ranking from implicit feedback”. In: *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*. AUAI Press. 2009, pp. 452–461.

[136] Steffen Rendle et al. “Fast context-aware recommendations with factorization machines”. In: *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*. ACM. 2011, pp. 635–644.

[137] Steffen Rendle et al. “Learning optimal ranking with tensor factorization for tag recommendation”. In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2009, pp. 727–736.

[138] Achim Rettinger et al. “Context-aware tensor decomposition for relation prediction in social networks”. In: *Social Network Analysis and Mining* 2.4 (2012), pp. 373–385.

[139] David A Ross et al. “Incremental learning for robust visual tracking”. In: *Int. J. Comput. Vis.* 77.1-3 (2008), pp. 125–141.

[140] Sujoy Roy and Sharat Chandra Guntuku. “Latent factor representations for cold-start video recommendation”. In: *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM. 2016, pp. 99–106.

[141] Paul Rozin and Edward B Royzman. “Negativity bias, negativity dominance, and contagion”. In: *Personality and social psychology review* 5.4 (2001), pp. 296–320.

[142] Alan Said and Alejandro Bellogín. “Comparative recommender system evaluation: benchmarking recommendation frameworks”. In: *Proceedings of the 8th ACM Conference on Recommender systems*. ACM. 2014, pp. 129–136.

[143] Ruslan Salakhutdinov and Andriy Mnih. “Bayesian probabilistic matrix factorization using Markov chain Monte Carlo”. In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 880–887.

[144] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. “Restricted Boltzmann machines for collaborative filtering”. In: *Proceedings of the 24th international conference on Machine learning*. ACM. 2007, pp. 791–798.

[145] Scott Sallinen et al. “High performance parallel stochastic gradient descent in shared memory”. In: *Parallel and Distributed Processing Symposium, 2016 IEEE International*. IEEE. 2016, pp. 873–882.

[146] Badrul Sarwar et al. *Application of dimensionality reduction in recommender system-a case study*. Tech. rep. Minnesota Univ Minneapolis Dept of Computer Science, 2000.

[147] Martin Saveski and Amin Mantrach. “Item cold-start recommendations: learning local collective embeddings”. In: *Proceedings of the 8th ACM Conference on Recommender systems*. ACM. 2014, pp. 89–96.

[148] J Ben Schafer et al. “Collaborative Filtering Recommender Systems”. In: *The Adaptive Web: Methods and Strategies of Web Personalization*. Ed. by Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl. Berlin, Heidelberg: Springer, 2007, pp. 291–324. ISBN: 978-3-540-72079-9.

[149] Sebastian Schelter, Venu Satuluri, and Reza Zadeh. *Factorbird-a parameter server approach to distributed matrix factorization*. 2014.

[150] Sebastian Schelter et al. “Distributed matrix factorization with mapreduce using a series of broadcast-joins”. In: *Proceedings of the 7th ACM conference on Recommender systems*. ACM. 2013, pp. 281–284.

[151] Tobias Schnabel et al. “Recommendations As Treatments: Debiasing Learning and Evaluation”. In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML’16. New York, NY, USA: JMLR.org, 2016, pp. 1670–1679.

[152] Lili Shan et al. “Predicting ad click-through rates via feature-based fully coupled interaction tensor factorization”. In: *Electronic Commerce Research and Applications* 16 (2016), pp. 30–42.

[153] Guy Shani and Asela Gunawardana. “Evaluating recommendation systems”. In: *Recommender systems handbook*. Springer, 2011, pp. 257–297.

[154] John Shawe-Taylor and Nello Cristianini. *Kernel methods for pattern analysis*. Vol. 47. Cambridge University Press, 2004, p. 462.

[155] Yue Shi, Martha Larson, and Alan Hanjalic. “Mining mood-specific movie similarity with matrix factorization for context-aware recommendation”. In: *Proceedings of the workshop on context-aware movie recommendation*. ACM. 2010, pp. 34–40.

[156] Yue Shi et al. “CLiMF: learning to maximize reciprocal rank with collaborative less-is-more filtering”. In: *Proceedings of the sixth ACM conference on Recommender systems*. ACM. 2012, pp. 139–146.

[157] Yue Shi et al. “TFMAP: optimizing MAP for top-n context-aware recommendation”. In: *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*. ACM. 2012, pp. 155–164.

[158] Yue Shi et al. “xCLiMF: optimizing expected reciprocal rank for data with multiple levels of relevance”. In: *Proceedings of the 7th ACM conference on Recommender systems*. ACM. 2013, pp. 431–434.

[159] Ajit P Singh and Geoffrey J Gordon. “Relational learning via collective matrix factorization”. In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2008, pp. 650–658.

[160] Nathan Srebro and Tommi Jaakkola. “Weighted low-rank approximations”. In: *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. 2003, pp. 720–727.

[161] Harald Steck. “Training and testing of recommender systems on data missing not at random”. In: *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2010, pp. 713–722.

[162] Gilbert Strang. *Linear Algebra and Its Applications*. 4th. Brooks Cole, 2006.

[163] Jian-Tao Sun et al. “Cubesvd: a novel approach to personalized web search”. In: *Proceedings of the 14th international conference on World Wide Web*. ACM. 2005, pp. 382–390.

[164] *Surprise. A Python scikit for recommender systems*. URL: `http://surpriselib.com` (visited on 09/12/2018).

[165] Panagiotis Symeonidis. “Clusthosvd: item recommendation by combining semantically enhanced tag clustering with tensor hosvd”. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 46.9 (2016), pp. 1240–1251.

[166] Panagiotis Symeonidis. “Content-based dimensionality reduction for recommender systems”. In: *Data Analysis, Machine Learning and Applications*. Springer, 2008, pp. 619–626.

[167] Panagiotis Symeonidis. “User recommendations based on tensor dimensionality reduction”. In: *IFIP International Conference on Artificial Intelligence Applications and Innovations*. Springer. 2009, pp. 331–340.

[168] Panagiotis Symeonidis, Alexandros Nanopoulos, and Yannis Manolopoulos. "A unified framework for providing recommendations in social tagging systems based on ternary semantic analysis". In: *IEEE Transactions on Knowledge and Data Engineering* 22.2 (2010), pp. 179–192.

[169] Panagiotis Symeonidis, Alexandros Nanopoulos, and Yannis Manolopoulos. "Tag recommendations based on tensor dimensionality reduction". In: *Proceedings of the 2008 ACM conference on Recommender systems*. ACM. 2008, pp. 43–50.

[170] Panagiotis Symeonidis et al. "Ternary Semantic Analysis of Social Tags for Personalized Music Recommendation." In: *Ismir*. Vol. 8. 2008, pp. 219–224.

[171] Gábor Takács, István Pilászy, and Domonkos Tikk. "Applications of the conjugate gradient method for implicit feedback collaborative filtering". In: *Proceedings of the fifth ACM conference on Recommender systems*. ACM. 2011, pp. 297–300.

[172] Gábor Takács and Domonkos Tikk. "Alternating least squares for personalized ranking". In: *Proceedings of the sixth ACM conference on Recommender systems*. ACM. 2012, pp. 83–90.

[173] Gábor Takács et al. "Investigation of various matrix factorization methods for large recommender systems". In: *Proceedings of the 2nd KDD Workshop on Large-Scale Recommender Systems and the Netflix Prize Competition*. ACM. 2008, p. 6.

[174] Gábor Takács et al. "Major components of the gravity recommendation system". In: *ACM SIGKDD Explorations Newsletter* 9.2 (2007), pp. 80–83.

[175] *Turi*. URL: `https://turi.com/` (visited on 09/12/2018).

[176] Madeleine Udell et al. "Generalized low rank models". In: *Foundations and Trends® in Machine Learning* 9.1 (2016), pp. 1–118.

[177] Bart Vandereycken. "Low-rank matrix completion by Riemannian optimization". In: *SIAM Journal on Optimization* 23.2 (2013), pp. 1214–1236.

[178] Koen Verstrepen et al. "Collaborative filtering for binary, positiveonly data". In: *ACM SIGKDD Explorations Newsletter* 19.1 (2017), pp. 1–21.

[179] Licai Wang, Xiangwu Meng, and Yujie Zhang. “Applying HOSVD to alleviate the sparsity problem in context-aware recommender systems”. In: *Chinese Journal of Electronics* 22.4 (2013), pp. 773–778.

[180] Zhijin Wang and Liang He. “User identification for enhancing IP-TV recommendation”. In: *Knowledge-Based Systems* 98 (2016), pp. 68–75.

[181] Markus Weimer et al. “Cofi rank-maximum margin matrix factorization for collaborative ranking”. In: *Advances in neural information processing systems*. 2008, pp. 1593–1600.

[182] Hendrik Wermser, Achim Rettinger, and Volker Tresp. “Modeling and learning context-aware recommendation scenarios using tensor decomposition”. In: *Advances in Social Networks Analysis and Mining (ASONAM), 2011 International Conference on*. IEEE. 2011, pp. 137–144.

[183] Jason Weston, Samy Bengio, and Nicolas Usunier. “Wsabie: Scaling up to large vocabulary image annotation”. In: *IJCAI*. Vol. 11. 2011, pp. 2764–2770.

[184] Wolfgang Woerndl and Johann Schlichter. “Introducing context into recommender systems”. In: *Proceedings of AAAI workshop on recommender systems in E-commerce*. 2007, pp. 138–140.

[185] Xin Xin et al. “FHSM: Factored Hybrid Similarity Methods for Top-N Recommender Systems”. In: *Asia-Pacific Web Conference*. Springer. 2016, pp. 98–110.

[186] Liang Xiong et al. “Temporal collaborative filtering with bayesian probabilistic tensor factorization”. In: *Proceedings of the 2010 SIAM International Conference on Data Mining*. SIAM. 2010, pp. 211–222.

[187] Yanfei Xu, Liang Zhang, and Wei Liu. “Cubic analysis of social bookmarking for personalized recommendation”. In: *Asia-Pacific Web Conference*. Springer. 2006, pp. 733–738.

[188] Jiyan Yang and Alex Gittens. *Tensor machines for learning target-specific polynomial features*. 2015. arXiv: 1504.01697. URL: https://arxiv.org/abs/1504.01697.

[189] Hsiang-Fu Yu, Mikhail Bilenko, and Chih-Jen Lin. “Selection of negative samples for one-class matrix factorization”. In: *Proceedings of the 2017 SIAM International Conference on Data Mining*. SIAM. 2017, pp. 363–371.

[190] Hsiang-Fu Yu et al. “Scalable coordinate descent approaches to parallel matrix factorization for recommender systems”. In: *Proceedings of the 2012 IEEE 12th International Conference on Data Mining*. ICDM ’12. IEEE. 2012, pp. 765–774.

[191] Shipeng Yu et al. “Collaborative ordinal regression”. In: *Proceedings of the 23rd international conference on Machine learning*. ACM. 2006, pp. 1089–1096.

[192] Hyokun Yun et al. “NOMAD: Non-locking, stOchastic Multi-machine algorithm for Asynchronous and Decentralized matrix completion”. In: *Proceedings of the VLDB Endowment* 7.11 (2014), pp. 975–986.

[193] Hongyuan Zha and Horst D Simon. “On updating problems in latent semantic indexing”. In: *SIAM Journal on Scientific Computing* 21.2 (1999), pp. 782–791.

[194] Hongyuan Zha and Zhenyue Zhang. “Matrices with low-rank-plus-shift structure: partial SVD and latent semantic indexing”. In: *SIAM Journal on Matrix Analysis and Applications* 21.2 (2000), pp. 522–536.

[195] Ce Zhang and Christopher Ré. “Dimmwitted: A study of main-memory statistical analytics”. In: *Proceedings of the VLDB Endowment* 7.12 (2014), pp. 1283–1294.

[196] Miao Zhang, Chris Ding, and Zhifang Liao. “Tensor fold-in algorithms for social tagging prediction”. In: *Data Mining (ICDM), 2011 IEEE 11th International Conference on*. IEEE. 2011, pp. 1254–1259.

[197] Wancai Zhang, Hailong Sun, Xudong Liu, et al. “An incremental tensor factorization approach for web service recommendation”. In: *Data Mining Workshop (ICDMW), 2014 IEEE International Conference on*. IEEE. 2014, pp. 346–351.

[198] Yongfeng Zhang et al. “Understanding the sparsity: Augmented matrix factorization with sampled constraints on unobservables”. In: *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. ACM. 2014, pp. 1189–1198.

[199] Xinqiang Zhao et al. “Crafting a time-aware point-of-interest recommendation via pairwise interaction tensor factorization”. In: *International Conference on Knowledge Science, Engineering and Management*. Springer. 2015, pp. 458–470.

[200] Guoxu Zhou et al. “Nonnegative matrix and tensor factorizations: An algorithmic perspective”. In: *IEEE Signal Processing Magazine* 31.3 (2014), pp. 54–65.

[201] Yunhong Zhou et al. “Large-scale parallel collaborative filtering for the netflix prize”. In: *Lecture Notes in Computer Science* 5034 (2008), pp. 337–348.

[202] Yong Zhuang et al. “A fast parallel SGD for matrix factorization in shared memory systems”. In: *Proceedings of the 7th ACM conference on Recommender systems*. ACM. 2013, pp. 249–256.