Chapter 7
Tag Recommendation

Tagging is an important feature of the Web 2.0. It allows the user to annotate items/ resources like songs, pictures, bookmarks, etc. with keywords. Tagging helps the user to organize his items and facilitate e.g. browsing and searching. Tag recommenders assist the tagging process of a user by suggesting him a set of tags that he is likely to use for an item. Personalized tag recommenders take the user's tagging behaviour in the past into account when they recommend tags. That means each user is recommended a personalized list of tags – i.e. the suggested list of tags depends both on the user and the item. Personalization makes sense as people tend to use different tags for tagging the same item. This can be seen in systems like Last.fm that have a non-personalized tag recommender but still the people use different tags. For this, we will provide an empirical evaluation on a subset of Last.fm that shows, that our proposed personalized tag recommender outperform even the theoretical upper-bound for any non-personalized tag recommender.

In this work, we apply our context-aware ranking framework to tag recommendation. In tag recommendation, the context is the user/item-pair for which tags should be recommended. E.g. when a user wants to annotate a bookmark, the system provides recommendations that are both suitable for the bookmark and the user. That means in contrast to item recommendation where the context was just the user and the items should be ranked (see chapter 6), now tags/ keywords should be ranked for a context that includes both a user and an item. In total, tag recommendation is a three-mode problem.

First, we will discuss the related work in the area of tag recommenders. Then, the problem of tag recommendation is analyzed in detail and the relationship to context-aware ranking is shown. Afterwards, we show how the BCR method can be applied to tag recommendation. Here, we discuss both the optimization criterion BCR-OPT and the learning algorithm BCR-LEARN. The factorization models of chapter 5 can be used to represent the latent interactions within the data. For tag recommendation, the TD model has a cubic runtime in the factorization dimensionality (i.e. $O(k^3)$) and thus it is slow even for mid-sized dimensionalities. In contrast, the PITF and PARAFAC models have linear runtime. Learning TD with other optimization strategies than bootstrapped stochastic gradient descent can lead to faster learning.

S. Rendle: Context-Aware Ranking with Factorization Models, SCI 330, pp. 85–111.
springerlink.com

We will discuss therefore the relationships to Higher-Order-Singular-Value-Decomposition (HOSVD) for tag recommendation (Symeonidis et al, 2008) which corresponds to a dense least-square optimization. Furthermore, we describe post-wise AUC optimization of the Tucker Decomposition, which was introduced by us as *Ranking Tensor Factorization* (RTF) (Rendle et al, 2009). In the evaluation, we compare the PITF and PARAFAC model learned by BCR to RTF, HOSVD (=least-square TD) and the non-factorization approaches Folkrank and adapted Pagerank. We will show, that our tensor factorization approaches RTF and BCR-PITF provide the best quality improving both FolkRank and PageRank. Furthermore, we show that factorization approaches have a better prediction runtime that FolkRank. Finally, our experiments indicate that our method BCR-PITF outperforms the best quality method RTF-TD largely in runtime as the runtime drops from $O(k^3)$ to $O(k)$ — where k is the factorization dimension.

Besides lab experiments, our factorization models using the BCR-based optimization provided the best results for the ECML/PKDD Discovery Challenge 2009 for graph-based tag recommendation. This challenge was won by our BCR-PITF model (Rendle and Schmidt-Thieme, 2009).

7.1 Related Work

Even though tagging is a new trend in the WWW, tag recommendation has already attracted much research. Next, we will discuss this research and categorize it in personalized and non-personalized recommenders.

7.1.1 Personalized Tag Recommendation

Personalized tag recommendation is a recent topic in recommender systems. FolkRank, an adaption of PageRank, was introduced by Hotho et al (2006). FolkRank generates high quality recommendations (Jäschke et al, 2008) outperforming several baselines like most-popular models and collaborative filtering (Jäschke et al, 2007). Even though FolkRank showed to provide high quality recommendations, due to its very slow prediction runtime it is not applicable for large real-world scenarios. In contrast to this, the prediction runtime of factorization models are independent of the number of users and items – after the model has been learned.

Recently, factorization models based on Tucker Decomposition (TD) have been introduced to tag recommendation. Symeonidis et al (2008) apply a Higher-Order-Singular-Value-Decomposition (HOSVD) (Lathauwer et al, 2000) for computing a low rank approximation of the tensor of the observed data. This approach corresponds to a TD model optimized for square-loss where all not observed values are learned as 0s (see section 4.3.2). Thus, these dense square-loss approaches like HOSVD or other least-square methods (Lathauwer et al, 2000b) do not lead to optimal factorizations for the task of tag recommendation as we will show both theoretically and empirically.

Various methods have been proposed for tag recommendation at the ECML/PKDD discovery challenge 2009 (DC09). We briefly describe the best performing methods. Marinho et al (2009) present a recommender method based on relational classification. A graph over posts is generated and tags are recommended based on the tags of the related neighbours. The paper also describes a semi-supervised extension that makes use of the posts that should be labeled in the future. For weighting the edges in the graph, several schemes based on cosine similarity are proposed. Zhang et al (2009) ensemble a collaborative filtering model based on user similarity with the FolkRank algorithm. The model in (Wetzker et al, 2009, 2010) is based on the assumption that different users have different vocabularies ('personomies'). Their approach is a linear combination between the most popular tags for an item and the estimated tags from the user's vocabulary to this item.

In contrast to the methods described so far, other approaches make use of additional content information. (Lipczak et al, 2009) propose an ensemble of six basic recommender which makes use of the text in the title and in URLs as well as resource related tags and the tags a user has given before. Similarly, Ju and Hwang (2009) extract candidates from textual content information of an item. They estimate the relevance of the candidates by their frequency and by matching them against the historical tags. This is combined with the tags the user has given in the past and the tags that were assigned to an item in the past. In section 7.6.3.1 our method is empirically compared these approaches on task 2 of the DC09.

7.1.2 Non-personalized Tag Recommendation

A non-personalized tag recommender predicts the same list of tags for the same item – i.e. it is independent of the user. There is several work on non-personalized tag recommenders, e.g. (Heymann et al, 2008; Song et al, 2008b,a). For example, Song et al (2008b) introduce an algorithm based on a Poisson mixture model. Although the algorithm is able to make predictions nearly in linear time, it is not personalized since the training data is composed from (words, documents, tags) triples containing no user specific information. Another difference to our work is that their method is content aware. Song et al (2008a) cast the problem of tag recommendation as a multi-label ranking problem for document classification and a fast recommendation algorithm based on gaussian processes is proposed. The algorithm provides linear time to train, proportional to the number of training samples, and constant time to predict per test case. Again differently from us, this approach is non-personalized since a given test document would be classified with the same set of tags independently of the users. Our evaluation (see section 7.6.3.1) indicates that if user information is present, our proposed personalized tag recommender outperforms any non-personalized tag recommender.

7.2 Personalized Tag Recommendation

Personalized tag recommendation is the task of recommending a list of tags to a user for annotating (e.g. describing) an item. An example is a music website where

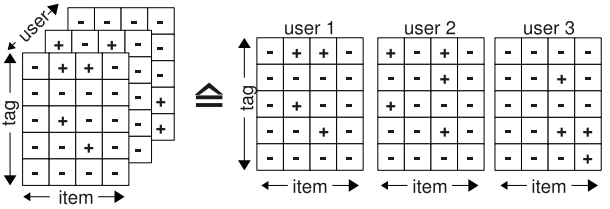


Fig. 7.1 The observed positive examples (u,i,t) are a ternary relationship that can be seen as a 3 dimensional tensor (cube). For each user a matrix is given that contains the tags given for a specific item.

a listener (user) wants to tag a song (item) and the system recommends him a list of keywords that the listener might want to use for this song. For inferring the recommendation list, a personalized tag recommender can use the historical data of the system, i.e. the tagging behaviour of the past. E.g. the recommender can make use of the tags that this user has given to other (similar) items in the past – or more general of similar tags that similar users haven given to similar items.

7.2.1 Formalization

Next, we show how tag recommendation can be expressed as a context-aware ranking problem of mode three ($m = 3$). For easier readability, we will use more meaningful names than X_i for the variable domains: $U = \{u_1, u_2, \ldots\}$ is the set of all users, $I = \{i_1, i_2, \ldots\}$ the set of all items and $T = \{t_1, t_2, \ldots\}$ the set of all tags. The historical tagging information of the past is given by $S \subseteq U \times I \times T$. As this is a ternary relation over categorical variables, it can be seen as a three-dimensional tensor (see figure 7.1) where the triples in S are the positive observations in the past. For tag recommendation, we are interested in recommending for a given user-item pair (u,i) a list of tags. That means, tag recommendation can be seen as context-aware ranking, where the context are the user-item pairs:

$$\mathscr{C} = U \times I \tag{7.1}$$

Following (Jäschke et al, 2007), we call each context $\mathbf{c} = (u,i) \in \mathscr{C}$ a *post* and we define the set of all observed posts P_S:

$$P_S := \{(u,i) | \exists t \in T : (u,i,t) \in S\}$$

P_S can be seen as a two-dimensional projection of S on the user/item dimension using the OR operation.

Now, the task of tag recommendation is the task of finding a context-aware ranking $\succ_{\mathbf{c}} = \succ_{u,i}$ for each post. Like suggested in section 3.4, all of the models presented here predict a scoring function $\hat{y} : U \times I \times T \to \mathbb{R}$ which can be used to derive a context-aware order according to eq. (3.34).

7.2.2 Data Analysis

The main problem in applying machine learning techniques on tagging system is that there are only observations S of positive tagging events (see figure 7.1). But it is unclear how the rest of this relation $(U \times I \times T) \setminus S$ should be interpreted.

7.2.2.1 0/1 Interpretation Scheme

A common interpretation scheme – we call it the *0/1 scheme* – is to encode positive feedback as 1 and interpret the remaining data as 0 (see figure 7.1). That means, the observed data is directly used for optimization. This interpretation is e.g. used for training tag recommenders using a HOSVD model (Symeonidis et al, 2008).

The 0/1 interpretation has three severe drawbacks:

1. The semantics are obviously incorrect. Imagine a user u has never tagged an item i before (e.g. figure 7.1, first item for user 1). For training a model with 0/1 interpretation all tags of this item are encoded with 0 and for learning the model is fitted to this data. So the model tries to predict a 0 for each case. The only reason why the model can predict something else than 0 is that it usually generalizes and does not fit exactly on the training data.
2. Also from a sparsity point of view the 0/1 scheme leads to a problem. If all elements that are not in S are assumed to be 0, even for a small dataset like Bibsonomy (see section 7.6.1), the 0 values dominate the 1 by many orders of magnitude. To give a practical example, first the sparsity for 0/1 interpretation is:

$$1 - \frac{|S|}{|U| \cdot |I| \cdot |T|} \tag{7.2}$$

   With this definition, for the BibSonomy 5-core dataset 99.94% elements are 0 and for the larger Last.fm 10-core dataset 99.998% are 0.
3. As one is interested in ranked lists, trying to fit to the numerical values of 1 and 0 is an unnecessary constraint. Instead only the qualitative difference between a positive and negative example is important. That means $\hat{y}$ of a positive example should be larger than that of a negative example.

7.2.2.2 Post-based Ranking Interpretation Scheme

Instead, we propose to infer pairwise ranking constraints D_S from S like we have discussed in section 3.3.1. The idea is that within a post (u,i), one can assume that a tag t_A is preferred over another tag t_B iff (u,i,t_A) has been observed and (u,i,t_B) has not been observed. An example is given in figure 7.2. In total, the training data D_S for pairwise constraints is defined as:

$$D_S := \{(u,i,t_A,t_B) : (u,i,t_A) \in S \wedge (u,i,t_B) \notin S\} \tag{7.3}$$

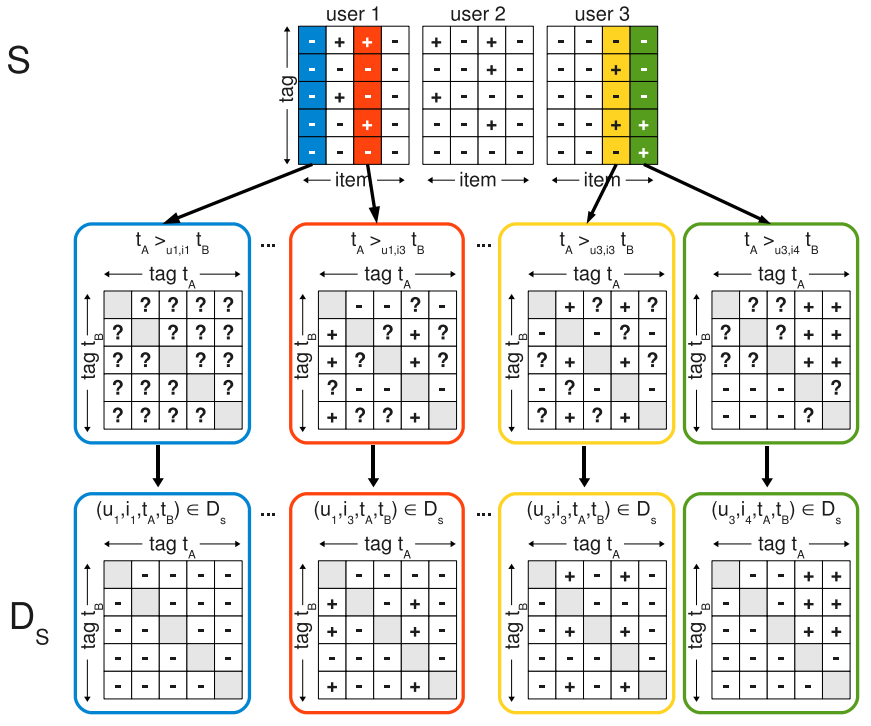


Fig. 7.2 Post-based ranking interpretation: From the observed data S, pairwise preferences D_S of tags can be inferred per post (user/ item combination). The figure shows examples for four posts: (u_1,i_1), (u_1,i_3), (u_3,i_3) and (u_3,i_4). E.g. for post (u_1,i_3), the following positive constraints can be inferred: $t_1 \succ_{u_1,i_3} t_2, t_1 \succ_{u_1,i_3} t_3, t_1 \succ_{u_1,i_3} t_5, t_4 \succ_{u_1,i_3} t_2, t_4 \succ_{u_1,i_3} t_3, t_4 \succ_{u_1,i_3} t_5$. For posts without any observed tag (like (u_1,i_1)), no constraints can be inferred.

The ranking relation $t_A \overset{?}{\succ}_{u,i} t_B$ between two non-observed tags ($(u,i,t_A) \notin S \wedge (u,i,t_B) \notin S$) is the one that should be predicted in the future. The main advantage of our approach is, that these pairs are treated as missing values (see the '?'s in figure 7.2, second row). Other approaches like (Symeonidis et al, 2008) learn that all these tags are not liked – i.e. they should have the same preference score 0. From a semantical point of view our scheme makes more sense as the user/ item combinations (posts) that have no tags (e.g. (u_1,i_1)), are the ones that the recommender system will have to predict a ranking for in the future. With our interpretation we treat this kind of data as missing values and do not use it as training data like in the '0/1 scheme'. Also inside a given post the non-observed tags are not fitted to 0, instead we only require that the positive examples have a higher value than the negative ones. This addresses the first two drawbacks of the '0/1 scheme'. The third drawback is tackled by our scheme by allowing free values for y and only posing pairwise ranking constraints (see eq. (3.34) and (7.3)). In all, a model for 'post-based ranking interpretation' should be optimized to satisfy as many ranking constraints as possible. In the following, we show how this can be done by the BCR method.

7.3 Bayesian Post-aware Ranking (BPoR) for Tag Recommendation

As we have shown in the last section, the problem of tag recommendation can be seen as an instance of context-aware ranking. Thus, the Bayesian Context-aware Ranking method gives the MAP estimator for the ranking $\succ$ that is defined by $\hat{y}$. Because the context is a post, we will refer to BCR for tag recommendation as *Bayesian Post-aware Ranking* (BPoR). Next, we show how to apply the optimization criterion BCR-OPT and the learning algorithm BCR-LEARN to tag recommenders.

*7.3.1 Optimization Criterion (*BPOR-OPT*)*

Like in section 4.1, the MAP estimator for the model parameters Θ that parametrize $\hat{y}$ is:

$$\underset{\Theta}{\operatorname{argmax}}\, p(\Theta \mid \succ) = \underset{\Theta}{\operatorname{argmax}}\, p(\succ \mid \Theta)\, p(\Theta) \tag{7.4}$$

where the probability for $\succ$ is defined over all context (i.e. combinations of users and items) and pairs of tags:

$$p(\succ \mid \Theta) = \prod_{(u,i,t_A,t_B) \in U \times I \times T^2} p(t_A \succ_{u,i} t_B \mid \Theta)^{2\,\delta(t_A \succ_{u,i} t_B)} \tag{7.5}$$

The probability of each quadruple (u,i,t_A,t_B) can be defined using the estimator $\hat{y}$ for $\succ$ (see section 4.1.3) and thus:

$$p(t_A \succ_{u,i} t_B \mid \Theta) := \sigma(\hat{y}(u,i,t_A) - \hat{y}(u,i,t_B)) \tag{7.6}$$

Again, gaussian priors are assumed over the model parameters:

$$p(\Theta) = \prod_{\theta \in \Theta} \sqrt{\frac{\lambda_\theta}{\pi}} \exp\left(-\lambda_\theta\, \theta^2\right) \tag{7.7}$$

Putting everything together, leads to BPoR-Opt for tag recomendation which is defined as the MAP estimator given the training data D_S:

$$\underset{\Theta}{\operatorname{argmax}}\, \text{BPOR-OPT} := \underset{\Theta}{\operatorname{argmax}} \sum_{(u,i,t_A,t_B) \in D_S} \ln \sigma(\hat{y}(u,i,t_A) - \hat{y}(u,i,t_B)) - \sum_{\theta \in \Theta} \frac{1}{2}\, \lambda_\theta\, \theta^2 \tag{7.8}$$

where λ_θ are model specific regularization parameters.

*7.3.2 Learning Algorithm (*BPOR-LEARN*)*

Secondly, we derive a learning algorithm to optimize the model parameters Θ of $\hat{y}$ for BPOR-OPT. In general, optimizing BPOR-OPT is time consuming, as D_S is very

large – the size of D_S is in $O(|S|\,|T|)$. E.g. for the examples of our evaluation section this would be about $3,299,006,344$ quadruples for the ECML/PKDD Discovery Challenge 09 and $449,290,590$ quadruples for our Last.fm subset. Thus computing the full gradients is very slow and normal gradient descent is not feasible. Also stochastic gradient descent where the quadruples are traversed in a sorted way like per post or per user will be slow – an example for this can be found in figure 4.1. Instead, BPoR-Learn draws quadruples by bootstrapping following the idea of BCR-Learn (see section 4.2.2).

Gradients

The gradient of BPoR-Opt given a case (u,i,t_A,t_B) with respect to a model parameter θ is:

$$\frac{\partial}{\partial\theta}\text{BPoR-Opt} = \delta_{u,i,t_A,t_B}\frac{\partial}{\partial\theta}(\hat{y}(u,i,t_A)-\hat{y}(u,i,t_B)) - \lambda_\theta\,\theta \tag{7.9}$$

with:

$$\delta_{u,i,t_A,t_B} := (1-\sigma(\hat{y}(u,i,t_A)-\hat{y}(u,i,t_B))) \tag{7.10}$$

That means, to apply BPoR-Learn to a given model $\hat{y}$, only the gradient $\frac{\partial}{\partial\theta}\hat{y}(u,i,t)$ has to be known. In the next section, we derive our factorization models and also show their gradients for optimization w.r.t. BPoR-Opt using BPoR-Learn.

Drawing of Cases

Like described before, the number of quadruples in D_S is huge. Thus, it is not feasible to explicitly enumerate all those quadruples. It is possible to draw cases (u,i,t_A,t_B) from D_S without enumerating them. The reason is, that D_S consists of all positive triples $(u,i,t_A)\in S$ combined with a negative example $(u,i,t_B)\notin S$. Therefore, first a triple (u,i,t_A) from S is drawn. From this a quadruple of D_S can be created by drawing a negative triple $(u,i,t_B)\notin S$. Such a negative triple can be drawn without enumerating them, by (1) drawing $t_B\in T$ and (2) rejecting it, if $(u,i,t_B)\in S$. This drawing scheme is effective because rejection is unlikely as most tags are not observed within a given post (u,i).

BPoR-Learn

Algorithm 3 shows the generic learning method BPoR-Learn for optimizing BPoR-Opt for tag recommendation. Analogously to BCR-Learn (algorithm 1) it first initializes the model parameters with random values. Then the parameters are learned iteratively by stochastic gradient descent. A case $(u,i,t_A,t_B)\in D_S$ is created using the drawing approach described above. Then the derivatives are computed and

Algorithm 3 BPoR-Learn

Input: training data S, learning rate α, regularization parameters λ_θ
Output: model parameters Θ

initialize Θ from $\mathcal{N}(0,\sigma^2)$
repeat
  draw (u,i,t_A) uniformly from S
  draw t_B from $\{t : (u,i,t) \notin S\}$
  $\delta_{u,i,t_A,t_B} \leftarrow (1-\sigma(\hat{y}(u,i,t_A)-\hat{y}(u,i,t_B)))$
  for $\theta \in \Theta$ **do**
    $\theta \leftarrow \theta + \alpha\left(\delta_{u,i,t_A,t_B}\frac{\partial}{\partial\theta}(\hat{y}(u,i,t_A)-\hat{y}(u,i,t_B)) - \lambda_\theta\,\theta\right)$
  end for
until convergence
return Θ

a small step of length α towards maximizing the quality is taken. This is repeated until a stopping criterion is met.

7.4 Factorization Models for Tag Recommendation

In the following, we apply three factorization models to tag recommendation: Tucker decomposition (TD), Parallel factor analysis (PARAFAC) and our pairwise interaction tensor factorization model (PITF) (see figure 7.3). We will show for each model how it can be learned with BPoR and the relationships to the other models. All of our factorization models predict a scoring function $\hat{y} : U \times I \times T \to \mathbb{R}$ which can be seen as a three-dimensional tensor Y where the value of entry (u,i,t) is the score $\hat{y}_{u,i,t}$.

7.4.1 Tucker Decomposition (TD)

Tucker decomposition (Tucker, 1966) factorizes a higher-order cube into a core tensor and one factor matrix for each dimensions.

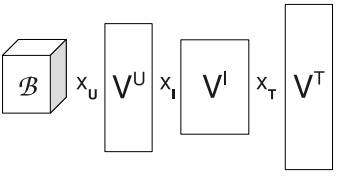
$$\hat{y}^{\mathrm{TD}}_{u,i,t} := \sum_{f_U=1}^{k_U}\sum_{f_I=1}^{k_I}\sum_{f_T=1}^{k_T} b_{f_U,f_I,f_T}\cdot v^U_{u,f_U}\cdot v^I_{i,f_I}\cdot v^T_{t,f_T} \tag{7.11}$$

or equivalently as tensor product (see figure 7.3):

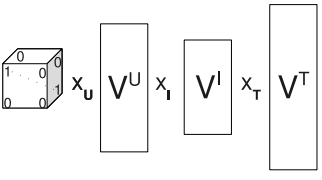
$$\hat{Y}^{\mathrm{TD}} := \mathscr{B}\times_U V^U \times_I V^I \times_T V^T \tag{7.12}$$

with model parameters:

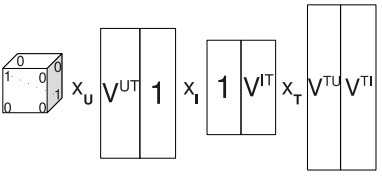
$$V^U \in \mathbb{R}^{|U|\times k_U},\quad V^I \in \mathbb{R}^{|I|\times k_I},\quad V^T \in \mathbb{R}^{|T|\times k_T},$$
$$\mathscr{B} \in \mathbb{R}^{k_U\times k_I\times k_T},\quad k_U,k_I,k_T \in \mathbb{N}^+ \tag{7.13}$$



(a) Tucker Decomposition



(b) PARAFAC



(c) Pairwise Interactions

Fig. 7.3 Tensor Factorization models: $\mathscr{B}$, V^U, V^I and V^T are the model parameters (one tensor, three matrices). In Tucker Decomposition the core tensor $\mathscr{B}$ is variable and the factorization dimensions can differ. For PARAFAC and Pairwise Interactions the core is a fixed diagonal tensor. In the Pairwise Interaction model, parts of the feature matrices are fixed which corresponds to modelling pairwise interactions.

For learning such a TD model with BPOR-OPT, the partial derivatives of $\hat{y}^{\text{TD}}$ given a case (u,i,t) are:

$$\frac{\partial \hat{y}^{\text{TD}}_{u,i,t}}{\partial b_{f_U,f_I,f_T}} = v^U_{u,f_U} \cdot v^I_{i,f_I} \cdot v^T_{t,f_T} \tag{7.14}$$

$$\frac{\partial \hat{y}^{\text{TD}}_{u,i,t}}{\partial v^U_{u,f_U}} = \sum_{f_I=1}^{k_I} \sum_{f_T=1}^{k_T} b_{f_U,f_I,f_T} \cdot v^I_{i,f_I} \cdot v^T_{t,f_T} \tag{7.15}$$

$$\frac{\partial \hat{y}^{\text{TD}}_{u,i,t}}{\partial v^I_{i,f_I}} = \sum_{f_U=1}^{k_U} \sum_{f_T=1}^{k_T} b_{f_U,f_I,f_T} \cdot v^U_{u,f_U} \cdot v^T_{t,f_T} \tag{7.16}$$

$$\frac{\partial \hat{y}^{\text{TD}}_{u,i,t}}{\partial v^T_{t,f_T}} = \sum_{f_U=1}^{k_U} \sum_{f_I=1}^{k_I} b_{f_U,f_I,f_T} \cdot v^U_{u,f_U} \cdot v^I_{i,f_I} \tag{7.17}$$

As discussed in section 5.1, the drawback of TD is that the model equation is a nested sum of degree 3 – i.e. it is cubic in $k := \min(k_u, k_i, k_t)$ and so the runtime complexity for predicting one triple (u,i,t) is in $O(k^3)$. Thus learning a TD model is slow even for a small to mid-sized number of factorization dimensions.

7.4.2 Parallel Factor Analysis (PARAFAC)

The PARAFAC model is a special case of the general Tucker decomposition model.

$$\hat{y}^{\text{PARAFAC}}_{u,i,t} := \sum_{f=1}^{k} v^{U}_{u,f} \cdot v^{I}_{i,f} \cdot v^{T}_{t,f} \tag{7.18}$$

As we have shown in section 5.2, PARAFAC can be derived from the Tucker decomposition model by setting $\mathscr{B}$ to the diagonal tensor.

The gradients of PARAFAC for tag recommendation are:

$$\frac{\partial \hat{y}^{\text{PARAFAC}}_{u,i,t}}{\partial v^{U}_{u,f}} = v^{I}_{i,f} \cdot v^{T}_{t,f}, \quad \frac{\partial \hat{y}^{\text{PARAFAC}}_{u,i,t}}{\partial v^{I}_{i,f}} = v^{U}_{u,f} \cdot v^{T}_{t,f}, \quad \frac{\partial \hat{y}^{\text{PARAFAC}}_{u,i,t}}{\partial v^{T}_{t,f}} = v^{U}_{u,f} \cdot v^{I}_{i,f} \tag{7.19}$$

Obviously, the PARAFAC model has a much better runtime complexity than TD because the model equation contains no nested sums and thus is in $O(k)$. A detailed discussion can be found in section 5.2.

7.4.3 Pairwise Interaction Tensor Factorization (PITF)

PITF explicitly models the two-way interactions between users, tags and items by factorizing each of the three relationships:

$$\hat{y}^{\text{PITF}}_{u,i,t} = \sum_{f=1}^{k_{U,T}} v^{U,T}_{u,f} \cdot v^{T,U}_{t,f} + \sum_{f=1}^{k_{I,T}} v^{I,T}_{i,f} \cdot v^{T,I}_{t,f} + \sum_{f=1}^{k_{U,I}} v^{U,I}_{u,f} \cdot v^{I,U}_{i,f} \tag{7.20}$$

The user-item interaction vanishes for predicting rankings and for BPoR optimization. The reason is that given a post (u,i), both the optimization criterion BPoR and the ranking ignores any score on the user-item interaction (see lemma 5.2). This results in the final model equation for PITF as follows:

$$\hat{y}^{\text{PITF}}_{u,i,t} = \sum_{f=1}^{k_{U,T}} v^{U,T}_{u,f} \cdot v^{T,U}_{t,f} + \sum_{f=1}^{k_{I,T}} v^{I,T}_{i,f} \cdot v^{T,I}_{t,f} \tag{7.21}$$

with model parameters:

$$V^{U,T} \in \mathbb{R}^{|U| \times k_{U,T}}, \quad V^{T,U} \in \mathbb{R}^{|T| \times k_{U,T}}, \quad k_{U,T} \in \mathbb{N}^{+} \tag{7.22}$$

$$V^{I,T} \in \mathbb{R}^{|I| \times k_{I,T}}, \quad V^{T,I} \in \mathbb{R}^{|T| \times k_{I,T}}, \quad k_{I,T} \in \mathbb{N}^{+} \tag{7.23}$$

As factorization dimensionality, we will always set $k = k_{U,T} = k_{I,T}$. In section 5.3, we have shown that PITF is a special case of PARAFAC and that the complexity of PITF is also in $O(k)$.

The gradients for the PITF model given a case (u,i,t) are:

$$\frac{\partial \hat{y}_{u,i,t}}{\partial v^{U,T}_{u,f}} = v^{T,U}_{t,f}, \quad \frac{\partial \hat{y}_{u,i,t}}{\partial v^{T,U}_{t,f}} = v^{U,T}_{u,f}, \quad \frac{\partial \hat{y}_{u,i,t}}{\partial v^{I,T}_{i,f}} = v^{T,I}_{t,f}, \quad \frac{\partial \hat{y}_{u,i,t}}{\partial v^{T,I}_{t,f}} = v^{I,T}_{i,f} \tag{7.24}$$

Algorithm 4 BPoR-LEARN-PITF

Input: training data S, learning rate α, regularization parameter λ
Output: model parameters $V^{U,T}, V^{T,U}, V^{I,T}, V^{T,I}$

initialize $V^{U,T}, V^{T,U}, V^{I,T}, V^{T,I}$ from $\mathcal{N}(0, \sigma^2)$
repeat
  draw (u, i, t_A) uniformly from S
  draw t_B from $\{t : (u, i, t) \notin S\}$
  $\delta_{u,i,t_A,t_B} \leftarrow (1 - \sigma(\hat{y}(u, i, t_A) - \hat{y}(u, i, t_B)))$
  for $f \in 1, \ldots, k$ **do**
    $v^{U,T}_{u,f} \leftarrow v^{U,T}_{u,f} + \alpha \left(\delta_{u,i,t_A,t_B} (v^{T,U}_{t_A,f} - v^{T,U}_{t_B,f}) - \lambda \theta \right)$
    $v^{I,T}_{i,f} \leftarrow v^{I,T}_{i,f} + \alpha \left(\delta_{u,i,t_A,t_B} (v^{T,I}_{t_A,f} - v^{T,I}_{t_B,f}) - \lambda \theta \right)$
    $v^{T,U}_{t_A,f} \leftarrow v^{T,U}_{t_A,f} + \alpha \left(\delta_{u,i,t_A,t_B} v^{U,T}_{u,f} - \lambda \theta \right)$
    $v^{T,U}_{t_B,f} \leftarrow v^{T,U}_{t_B,f} + \alpha \left(-\delta_{u,i,t_A,t_B} v^{U,T}_{u,f} - \lambda \theta \right)$
    $v^{T,I}_{t_A,f} \leftarrow v^{T,I}_{t_A,f} + \alpha \left(\delta_{u,i,t_A,t_B} v^{I,T}_{i,f} - \lambda \theta \right)$
    $v^{T,I}_{t_B,f} \leftarrow v^{T,I}_{t_B,f} + \alpha \left(-\delta_{u,i,t_A,t_B} v^{I,T}_{i,f} - \lambda \theta \right)$
  end for
until convergence
return $V^{U,T}, V^{T,U}, V^{I,T}, V^{T,I}$

Algorithm 4 shows the complete BPoR optimization method for PITF.

7.4.4 Relation between TD, PARAFAC and PITF

In chapter 5, we have shown that for $m > 2$ (note in tag recommendation $m = 3$), the class of all PARAFAC models is a true subclass of TD and PITF is a true subclass of PARAFAC. I.e. here:

$$\mathcal{M}^{\text{TD}} \supset \mathcal{M}^{\text{PARAFAC}} \supset \mathcal{M}^{\text{PITF}} \tag{7.25}$$

From the perspective of expressiveness and complexity analysis, it does not make sense to use the PITF model. But in a sparse setting with little training data, it makes sense to use less expressive models with a fixed instead of a variable structure if this structure is carefully chosen. In our evaluation section, we empirically show that PITF models outperform PARAFAC models for tag recommendation when using Gaussian priors on the free model parameters like in eq. (7.7).

7.5 Alternative Optimization for Tucker Decomposition

Next, we want to discuss two alternative optimization criteria for TD models. The first one is HOSVD which corresponds to a dense least-square optimization of the TD parameters. The second one is RTF, a ranking optimization based on post-wise AUC maximization.

7.5.1 Higher-Order Singular Value Decomposition (HOSVD)

Singular Value Decomposition (SVD) is a well studied factorization method for two mode problems. It creates the best k-rank approximation of a matrix with respect to minimal least-square error. Higher-order singular value decomposition (HOSVD) is an extension of SVD to problems of higher mode ($m \geq 3$). The model equation of HOSVD is the Tucker decomposition model. But using (HO)SVD always implies that the optimization criterion is least square. Moreover (HO)SVD does not handle any missing values, i.e. the least square of all elements in the tensor is calculated. The optimization criterion for HOSVD for tag recommendation is:

$$\underset{\mathscr{B},V^U,V^I,V^T}{\operatorname{argmin}} \sum_{(u,i,t)\in(U\times I\times T)} \left(\hat{y}_{u,i,t} - \delta((u,i,t)\in S)\right)^2 \tag{7.26}$$

This corresponds to the 0/1 interpretation scheme, that we have discussed in section 7.2.2.1.

An advantage of HOSVD is that there are fast optimization algorithms for sparse settings – here sparsity means many 0 values. Lathauwer et al (2000) introduce an approximation that first generates three two-mode problems by unfolding the tensor. Then each two-mode problem is solved with a sparse SVD solver. The factorization matrices of the two-mode problems can be used as factorization matrices for the original problem. Finally, the core tensor can be calculated from the factorization matrices and the data.

In our evaluation, we will compare this approach to our proposed BPoR optimization.

7.5.2 Optimizing the Ranking Statistic AUC per Post (RTF)

In (Rendle et al, 2009), we have developed an improvement of HOSVD for the task of tag recommendation that optimizes the AUC on the observed posts. The method was named RTF for 'Ranking with Tensor Factorization'. In the following, we will describe this method and show the similarities and differences to BCR.

7.5.2.1 Optimization Criterion

RTF uses the same pairwise data interpretation D_S as in eq. (7.3). But the optimization is done with respect to the AUC over each observed post instead of BCR:

$$\underset{\mathscr{B},V^U,V^I,V^T}{\operatorname{argmax}} \sum_{(u,i)\in P_S} \operatorname{AUC}(u,i) \tag{7.27}$$

with

$$\operatorname{AUC}(u,i) := \frac{1}{|T^+_{u,i}||T^-_{u,i}|} \sum_{t_A\in T^+_{u,i}} \sum_{t_B\in T^-_{u,i}} \delta(\hat{y}_{u,i,t_A} > \hat{y}_{u,i,t_B}) \tag{7.28}$$

where:

$$T^+_{u,i} := \{t : (u,i,t) \in S\}, \quad T^-_{u,i} := T \setminus T^+_{u,i} \tag{7.29}$$

In section 4.3.1, we have shown that eq. (7.27) can be rewritten as:

$$\underset{\mathscr{B},V^U,V^I,V^T}{\operatorname{argmax}} \sum_{(u,i,t_A,t_B)\in D_S} z_{u,i}\,\delta(\hat{y}_{u,i,t_A} > \hat{y}_{u,i,t_B}) \tag{7.30}$$

where $z_{u,i}$ is the normalization constant:

$$z_{u,i} = \frac{1}{|T^+_{u,i}||T^-_{u,i}|} \tag{7.31}$$

Regularization

The optimization criterion presented so far will lead to the best value given the training data. With high feature dimensions (i.e. high k_U, k_I, k_T) an arbitrary small error on the training data can be achieved. In general we are not interested in a low error for the already observed data but in a low error over unseen data. Minimizing the training error for models with a large number of parameters will lead to overfitting, i.e. a small training error but a large error over new/ unseen data. A common way to prevent this is to regularize the optimization criterion. Regularization is very successful in related areas like rating prediction (Rennie and Srebro, 2005). Adding a regularization objective to the optimization task in formula (7.27) leads to the following objective:

$$\underset{\mathscr{B},V^U,V^I,V^T}{\operatorname{argmax}} \sum_{(u,i)\in P_S} \mathrm{AUC}(u,i) - \frac{1}{2}\left(\lambda_{\mathscr{B}}||\mathscr{B}||^2_F + \lambda_U||V^U||^2_F + \lambda_I||V^I||^2_F + \lambda_T||V^T||^2_F\right) \tag{7.32}$$

Where $\lambda_U, \lambda_I, \lambda_T, \lambda_{\mathscr{B}} \in \mathbb{R}^+_0$ are the regularization parameters for the the feature matrices and core tensor respectively.

7.5.2.2 Learning Algorithm

For optimizing eq. (7.27), the δ-function is approximated by the differentiable σ-function. As the AUC per observed post $(u,i) \in P_S$ should be optimized, we perform gradient descent per post, i.e. $\frac{\partial}{\partial\theta}\mathrm{AUC}(u,i)$ (see algorithm 5).

Gradients

The gradients for each model parameter given a post (u,i) are:

Algorithm 5 RTF-LEARN

Input: training data S, learning rate α, regularization parameters $\lambda_{\mathscr{B}}, \lambda_U, \lambda_I, \lambda_T$
Output: model parameters $\mathscr{B}, V^U, V^I, V^T$

initialize $\mathscr{B}, V^U, V^I, V^T$ from $\mathscr{N}(0, \sigma^2)$
repeat
  for $(u,i) \in P_S$ **do**
    for $(f_U, f_I, f_T) \in k_U \times k_I \times k_T$ **do**
      $b_{f_U,f_I,f_T} \leftarrow b_{f_U,f_I,f_T} + \alpha \left(\frac{\partial}{\partial b_{f_U,f_I,f_T}} \mathrm{AUC}(u,i) - \lambda_{\mathscr{B}} b_{f_U,f_I,f_T} \right)$
    end for
    for $f_U \in \{1,\ldots,k_U\}$ **do**
      $v^U_{u,f_U} \leftarrow v^U_{u,f_U} + \alpha \left(\frac{\partial}{\partial v^U_{u,f_U}} \mathrm{AUC}(u,i) - \lambda_U v^U_{u,f_U} \right)$
    end for
    for $f_I \in \{1,\ldots,k_I\}$ **do**
      $v^I_{i,f_I} \leftarrow v^I_{i,f_I} + \alpha \left(\frac{\partial}{\partial v^I_{i,f_I}} \mathrm{AUC}(u,i) - \lambda_I v^I_{i,f_I} \right)$
    end for
    for $t \in T$ **do**
      for $f_T \in \{1,\ldots,k_T\}$ **do**
        $v^T_{t,f_T} \leftarrow v^T_{t,f_T} + \alpha \left(\frac{\partial}{\partial v^T_{t,f_T}} \mathrm{AUC}(u,i) - \lambda_T v^T_{t,f_T} \right)$
      end for
    end for
  end for
until convergence
return $\mathscr{B}, V^U, V^I, V^T$

$$\frac{\partial}{\partial b_{f_U,f_I,f_T}} \mathrm{AUC}(u,i) = z_{u,i} \sum_{t_A \in T^+_{u,i}} \sum_{t_B \in T^-_{u,i}} \delta_{u,i,t_A,t_B} v^U_{u,f_U} v^I_{i,f_I} (v^T_{t_A,f_T} - v^T_{t_B,f_T}) \tag{7.33}$$

$$\frac{\partial}{\partial v^U_{u,f_U}} \mathrm{AUC}(u,i) = z_{u,i} \sum_{t_A \in T^+_{u,i}} \sum_{t_B \in T^-_{u,i}} \sum_{f_I=1}^{k_I} \sum_{f_T=1}^{k_T} \delta_{u,i,t_A,t_B} b_{f_U,f_I,f_T} v^I_{i,f_I} (v^T_{t_A,f_T} - v^T_{t_B,f_T}) \tag{7.34}$$

$$\frac{\partial}{\partial v^I_{i,f_I}} \mathrm{AUC}(u,i) = z_{u,i} \sum_{t_A \in T^+_{u,i}} \sum_{t_B \in T^-_{u,i}} \sum_{f_U=1}^{k_U} \sum_{f_T=1}^{k_T} \delta_{u,i,t_A,t_B} b_{f_U,f_I,f_T} v^U_{u,f_U} (v^T_{t_A,f_T} - v^T_{t_B,f_T}) \tag{7.35}$$

with:

$$\delta_{u,i,t_A,t_B} := \sigma(\hat{y}_{u,i,t_A} - \hat{y}_{u,i,t_B}) (1 - \sigma(\hat{y}_{u,i,t_A} - \hat{y}_{u,i,t_B})) \tag{7.36}$$

The gradients for the tags depend on whether the tag t is observed $t = t_A \in T^+_{u,i}$ or not $t = t_B \in T^-_{u,i}$:

$$\frac{\partial}{\partial v^T_{t_A,f_I}}\mathrm{AUC}(u,i) = z_{u,i} \sum_{t_B \in T^-_{u,i}} \sum_{f_U=1}^{k_U} \sum_{f_I=1}^{k_I} \delta_{u,i,t_A,t_B}\, b_{f_U,f_I,f_T}\, v^U_{u,f_U}\, v^I_{i,f_I} \tag{7.37}$$

$$\frac{\partial}{\partial v^T_{t_B,f_I}}\mathrm{AUC}(u,i) = -z_{u,i} \sum_{t_A \in T^+_{u,i}} \sum_{f_U=1}^{k_U} \sum_{f_I=1}^{k_I} \delta_{u,i,t_A,t_B}\, b_{f_U,f_I,f_T}\, v^U_{u,f_U}\, v^I_{i,f_I} \tag{7.38}$$

Fast Computation of Gradients

By storing intermediate results, these gradients can be calculated more efficiently. For the updates on the user, item and core, we precalculate the vector $\gamma \in \mathbb{R}^{k_T}$:

$$\gamma_{f_T} := \sum_{t^+ \in T^+_{u,i}} \sum_{t^- \in T^-_{u,i}} \delta_{u,i,t_A,t_B}(v^T_{t_A,f_T} - v^T_{t_B,f_T}) \tag{7.39}$$

The computation of the whole vector γ can be performed in $O(k_T\,|T^+|\,|T^-|)$. Now the gradients for the user, item and the core simplify to:

$$\frac{\partial}{\partial b_{f_U,f_I,f_T}}\mathrm{AUC}(u,i) = z_{u,i}\gamma_{f_T}\, v^U_{u,f_U}\, v^I_{i,f_I} \tag{7.40}$$

$$\frac{\partial}{\partial v^U_{u,f_U}}\mathrm{AUC}(u,i) = z_{u,i} \sum_{f_I=1}^{k_I} \sum_{f_T=1}^{k_T} \gamma_{f_T}\, b_{f_U,f_I,f_T}\, v^I_{i,f_I} \tag{7.41}$$

$$\frac{\partial}{\partial v^I_{i,f_I}}\mathrm{AUC}(u,i) = z_{u,i} \sum_{f_U=1}^{k_u} \sum_{f_T=1}^{k_T} \gamma_{f_T}\, b_{f_U,f_I,f_T}\, v^U_{u,f_U} \tag{7.42}$$

$$\tag{7.43}$$

With this, updating the whole core tensor ($k_U \cdot k_I \cdot k_T$ factors) and updating the k_U factors of user u and k_I factors of item i is in $O(k_T\,|T^+|\,|T^-| + k_U\,k_I\,k_T)$.

Similarly, for the tags the update can be simplified by storing the vector $\eta \in \mathbb{R}^{k_T}$:

$$\eta_{f_T} := \sum_{f_U=1}^{k_U} \sum_{f_I=1}^{k_I} b_{f_U,f_I,f_T}\, v^U_{u,f_U}\, v^U_{u,f_U} \tag{7.44}$$

which can be computed for the whole vector in $O(k_U\,k_I\,k_T)$. Now the gradients for the tags simplify to:

$$\frac{\partial}{\partial v^T_{t_A,f_I}}\mathrm{AUC}(u,i) = z_{u,i} \sum_{t_B \in T^-_{u,i}} \delta_{u,i,t_A,t_B}\, \eta_{f_T} \tag{7.45}$$

$$\frac{\partial}{\partial v^T_{t_B,f_I}}\mathrm{AUC}(u,i) = -z_{u,i} \sum_{t_A \in T^+_{u,i}} \delta_{u,i,t_A,t_B}\, \eta_{f_T} \tag{7.46}$$

To update the k_T factors for each of the $|T^+_{u,i}|$ tags that are observed, the complexity is $O(k_U\, k_I\, k_T + |T^+|\, k_T\, |T^-|)$. Similarly, for the $|T^+_{u,i}|$ negative tags, the complexity is $O(k_U\, k_I\, k_T + |T^-|\, k_T\, |T^+|)$.

In total, a gradient descent step for a whole post can be performed in $O(k_U\, k_I\, k_T + k_T\, |T^+|\, |T^-|)$. Mostly, the number of tags given $|T^+_{u,i}|$ is constant and independent of $|T|$ because the tags a user gives does not depend on how many tags (words) there are in total. Furthermore, $|T^+_{u,i}|$ is typically small. Under these assumptions, the complexity is in $O(k_U\, k_I\, k_T + k_T\, |T|)$.

Fast Prediction with TD Models

For predicting $\hat{y}_{u,i,t}$ with a TD model, formula (7.11) is used. The runtime complexity for eq. (7.11) is $O(k_U \cdot k_I \cdot k_T)$ and thus the trivial upper bound of the runtime for predicting a top-n list is $O(|T| \cdot k_U \cdot k_I \cdot k_T)$. Though the runtime can be improved largely by reordering the sums in eq. (7.11):

$$\begin{aligned}
\hat{y}_{u,i,t} &= \sum_{f_U=1}^{k_U} \sum_{f_I=1}^{k_I} \sum_{f_T=1}^{k_T} b_{f_U,f_I,f_T}\, v^U_{u,f_U}\, v^I_{i,f_I}\, v^T_{t,f_T} \\
&= \sum_{f_T=1}^{k_T} v^T_{t,f_T} \sum_{f_U=1}^{k_U} \sum_{f_I=1}^{k_I} b_{f_U,f_I,f_T}\, v^U_{u,f_U}\, v^I_{i,f_I} \\
&= \sum_{f_T=1}^{k_T} v^T_{t,f_T}\, \rho_{f_T} \qquad (7.47)
\end{aligned}$$

with:

$$\rho_{f_T} := \sum_{f_U=1}^{k_U} \sum_{f_I=1}^{k_I} b_{f_U,f_I,f_T}\, v^U_{u,f_U}\, v^I_{i,f_I} \qquad (7.48)$$

When making a top-n prediction for user u and item i instead of computing (7.11) for each tag, the intermediate result vector ρ can be computed first in $O(k_U \cdot k_I \cdot k_T)$. The top-n prediction can then be made using this intermediate result and the total runtime of predicting top-n is then $O(|T| \cdot k_T + k_U \cdot k_I \cdot k_T)$. Thus the runtime for prediction with the TD model is independent of the number of users, items and observations S. It only depends on the dimensions of the factorization and the number of tags.

7.5.2.3 Comparison to BPoR

As we have described in section 4.3.1, the optimization criterion of AUC is related to BPoR. The difference is the normalization constant $z_{u,i}$ of posts and that as error measure, σ (for AUC) instead of $\ln \sigma$ (for BPoR) is used. But RTF also differs from BPoR in terms of the learning algorithm. For RTF, the post-wise AUC is directly optimized by performing gradient descent on posts (u,i) whereas for BPoR

quadruples (u, i, t_A, t_B) are drawn. We will compare the empirical quality of BPoR, RTF and HOSVD next.

7.6 Evaluation

In our evaluation, we investigate the learning runtime and prediction quality of our proposed methods. For the runtime, we want to justify the results of the theoretical complexity analysis (TD is in O(k^3), PARAFAC/PITF in O(k)) by an empirical comparison of the TD model to the PARAFAC model and our PITF model. With respect to prediction quality, we investigate empirically whether the speedup of PARAFAC/ PITF is paid with quality – i.e. if there is a trade-off between quality and runtime between the model classes.

7.6.1 Datasets

We use three datasets for evaluation: Bibsonomy and Last.fm like in (Jäschke et al, 2007; Rendle et al, 2009) and the dataset from the ECML/ PKDD Discovery Challenge 2009[1]. All datasets are p-cores[2] – for BibSonomy the 5-core, for Last.fm the 10-core and for the ECML/PKDD Challenge the provided 2-core. The characteristics of the datasets can be found in table 7.1.

Table 7.1 Dataset characteristics in terms of number of users, items, tags, tagging triples S and posts.

Dataset	Users $\|U\|$	Items $\|I\|$	Tags $\|T\|$	Triples $\|S\|$	Posts $\|P_S\|$
BibSonomy	116	361	412	10,148	2,522
Last.fm	2,917	1,853	2,045	219,702	75,565
ECML/PKDD DC 09	1,185	22,389	13,276	248,494	63,628

7.6.2 Evaluation Methodology

For Bibsonomy and Last.fm we use the same protocol as described in (Jäschke et al, 2008) – i.e. per user one post is randomly removed from the training set S_{train} and put into the test set S_{test}. We use exactly the same splits as in (Jäschke et al, 2008). After the splits have been built, the recommenders are trained on the test set and then the prediction quality on the test set is measured. As quality measure, we report the F-Measure on top-N lists (see section 3.5). The experiments are repeated 10 times by sampling new training/ test sets. We report the average over all runs.

For the ECML Challenge dataset, we use the protocol and split of the challenge and report the official results.

[1] `http://www.kde.cs.uni-kassel.de/ws/dc09`

[2] The p-core of S is the largest subset of S with the property that every user, every item and every tag has to occur at least p times.

Hyperparameters

The hyperparameters of all models are searched on the first training split. For the RTF-TD model, the hyperparameters are: learning rate $\alpha = 0.5$ for BibSonomy and $\alpha = 0.1$ for Last.fm; regularization $\gamma = \gamma_c = 10^{-5}$ for BibSonomy and $\gamma = \gamma_c = 10^{-6}$ for Last.fm; iterations $iter = 500$ for BibSonomy and $iter = 600$ for Last.fm. The model parameters Θ are initialized with small random values drawn from the normal distribution $N(0, 0.1)$. For HOSVD we have a dimensionality of $(k_U, k_I, k_T) = (60, 105, 225)$ for BibSonomy and $(k_U, k_I, k_T) = (875, 556, 614)$ for Last.fm. For PITF the hyperparameters are $\lambda = 5e-05$ and $\alpha = 0.05$. For PARAFAC they are $\lambda = 0$ and $\alpha = 0.01$. The parameters of both PITF and PARAFAC are initialized with $N(0, 0.01)$. For FolkRank and PageRank, we report the values obtained by Jäschke et al (2008) as we use the same datasets and splits.

Implementations

The learning runtime measurements of RTF-TD, BPoR-PITF and BPoR-PARAFAC were made with C++ implementations. The runtime measurement for predicting is made with a Object Pascal implementation of RTF-TD and a C++ implementation of Folkrank. In general, the experiments were run on a compute cluster with 200 cores in total. Each compute node has identical hard- and software. Our implementations use no parallelization neither over compute nodes nor within nodes – i.e. per run only one processor core was used.

7.6.3 Results

We compare our factorization models BPoR-PITF, BPoR-PARAFAC and RTF-TD to other state-of-the-art personalized tag recommender and the upper bound for non-personalized tag-recommender. We investigate both the quality of the predictions and the runtime for learning and predicting.

7.6.3.1 Prediction Quality

First of all, we compare the prediction quality of our factorization models BPoR-PITF, BPoR-PARAFAC and RTF-TD to competing models. In figure 7.4, a comparison to FolkRank, PageRank and HOSVD on Bibsonomy and Last.fm is shown. In general, the factorization models result in the best prediction quality – only on the very small Bibsonomy dataset FolkRank is competitive.

PITF vs. PARAFAC

When comparing the two factorization models with linear runtime in k – i.e. PARAFAC and PITF – one can see that BPoR-PITF achieves on all datasets a higher

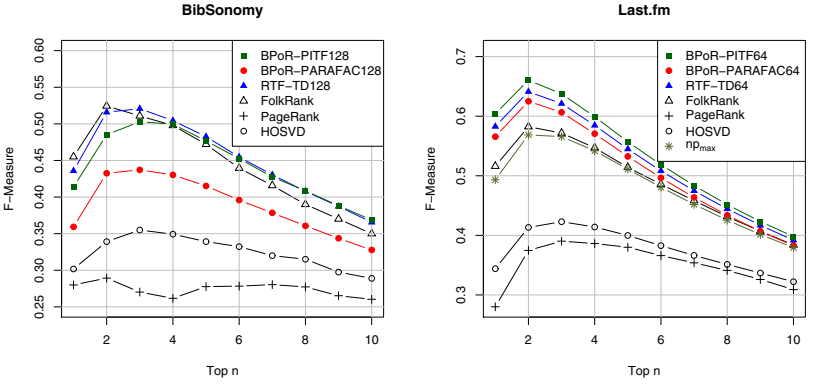


Fig. 7.4 Our factorization models (RTF-TD, BPoR-PARAFAC, BPoR-PITF) achieve the best prediction quality outperforming other approaches like FolkRank, PageRank and HOSVD. On the larger Last.fm dataset the BPoR-PITF model has the highest quality.

prediction quality than BPoR-PARAFAC. At first, this might be surprising because PARAFAC is more general and includes PITF. But it seems that BPoR-PARAFAC is unable to find the pairwise structure of PITF and to do regularization at the same time. An indication for this is that for PARAFAC the 'best' regularization parameter found by grid search is $\lambda = 0$.

PITF vs. RTF-TD

Next, we compare the prediction quality of the pairwise interaction model (BPoR-PITF) to full Tucker decomposition (RTF-TD) (see figure 7.4 and 7.5). On the small Bibsonomy dataset, on small top-N lists (1,2,3) RTF-TD outperforms BPoR-PITF whereas on larger lists, the difference vanishes. In contrast to this on the larger Last.fm dataset BPoR-PITF outperforms RTF-TD on all list sizes. These results indicate that the learning speedup of BPoR-PITF models to RTF-TD does not come to the prize of lower prediction quality. Rather, BPoR-PITF can even outperform RTF-TD in quality on larger datasets.

RTF-TD vs. HOSVD

The prediction quality of RTF-TD is clearly superior to the one of HOSVD. On BibSonomy even with a very small number of 8 dimensions, RTF-TD achieves almost similar results as HOSVD with a dimensionality of $(60, 105, 225)$ and $(875, 556, 614)$ for Last.fm respectively. Increasing the dimensions of RTF to 16 dimensions already largely outperforms HOSVD in quality. Note that for Last.fm this means that for HOSVD there are $298,711,000$ parameters to learn in the core tensor – whereas for RTF8 there are only 512 and for RTF16 only $4,096$ parameters.

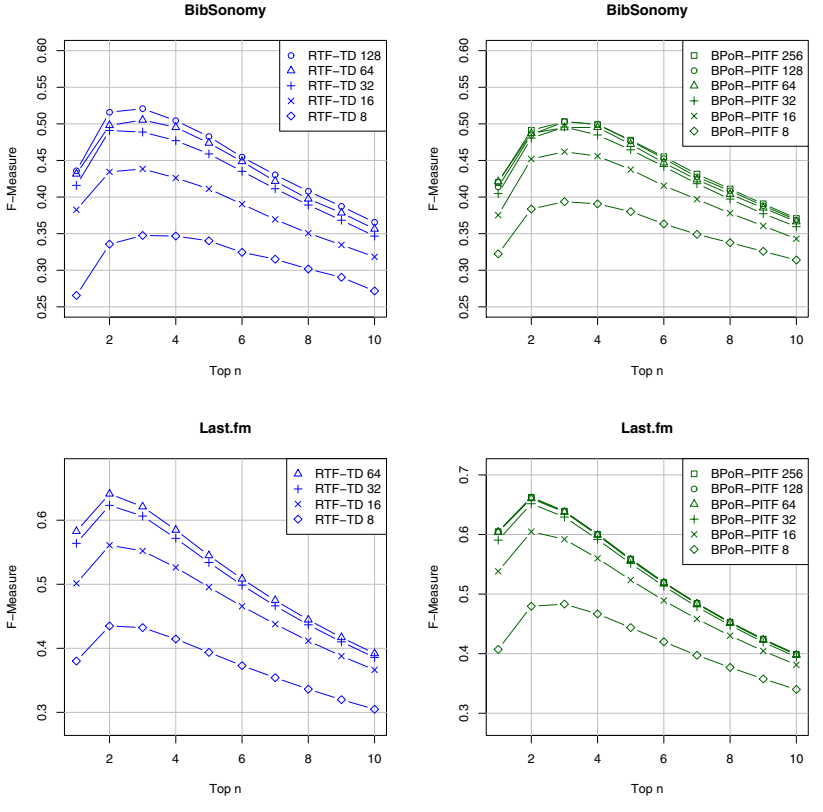


Fig. 7.5 PITF vs. RTF-TD for a varying number of factorization dimensions (k).

The empirical qualitative results match our discussion about the data interpretation in section 7.2.2.1.

Even though RTF-TD and HOSVD have the same prediction method (the Tucker decomposition) and thus prediction complexity, in practice RTF-TD models are much faster in prediction than comparable HOSVD models, because RTF-TD models need much less dimensions than HOSVD for achieving better quality. A final problem with HOSVD is that we found it to be very sensitive for the number of dimensions and that they have to be chosen carefully. Also HOSVD is sensitive to the relations between the user, item and tag dimensions – e.g. choosing the same dimension for all three dimensions leads to poor results. In contrast to this, for RTF-TD we can choose the same number of dimensions for user, item and tags. Furthermore for RTF-TD, by increasing the number of dimensions we get better results. We expect this behaviour due to the regularization of RTF-TD models.

Table 7.2 Official results (top-6) from the ECML/ PKDD Discovery Challenge 2009.

Rank	Method	Top-5 F-Measure
1	BPoR-PITF + adaptive list size	0.35594
-	*BPoR-PITF (not submitted)*	*0.345*
2	Relational Classification (Marinho et al, 2009)	0.33185
3	Content-based (Lipczak et al, 2009)	0.32461
4	Collaborative Filtering + FolkRank (Zhang et al, 2009)	0.32230
5	Content-based (Ju and Hwang, 2009)	0.32134
6	Personomy translation (Wetzker et al, 2009)	0.32124
...	...	...

Non-personalized Recommenders

In a last experiment, we compare the prediction quality of personalized tag recommenders to the best possible non-personalized tag recommender, i.e. the theoretical upper bound for **n**on-**p**ersonalized tag recommendation $\text{np}_{\max}$ (see figure 7.4). The weighting method for $\text{np}_{\max}$ is:

$$\hat{y}^{\text{np}_{\max}}_{u,i,t} := |\{u' | (u', i, t) \in S_{test}\}| \tag{7.49}$$

That means, for each item the tags are ranked by counting how often a tag appears in the *test* set. Please note that in practice $\hat{y}^{\text{np}_{\max}}$ cannot be applied as S_{test} is unknown. But here we use $\hat{y}^{\text{np}_{\max}}$ as the theoretical upper bound for non-personalized recommenders because it creates the best non-personalized top-n list for the test set S_{test} – every other method for non-personalized tag recommendation like Heymann et al (2008); Song et al (2008b,a) is guaranteed to have a lower (or in the best case the same) quality on S_{test}. As figure 7.4 shows, personalized tag recommenders like FolkRank, PITF and RTF outperform $\text{np}_{\max}$ the theoretical upper bound for non-personalized tag recommendation[3]. That means, in applications where personalized information is present, personalized tag recommenders are supposed to outperform non-personalized tag recommenders.

ECML / PKDD Discovery Challenge 09

In addition to the lab experiments, our BPoR-PITF model took also part in task 2 of the ECML/PKDD Discovery Challenge 09 and achieved the highest prediction quality. Table 7.2 shows the final results[4] listing the first six approaches. This evaluation in a tag recommender challenge organized by a third party shows that BPoR-PITF is able to create high quality predictions.

[3] Evaluating $\text{np}_{\max}$ on the small BibSonomy dataset makes no sense because in the test sets S_{test} of BibSonomy are rarely two posts with the same item.

[4] `http://www.kde.cs.uni-kassel.de/ws/dc09/results`

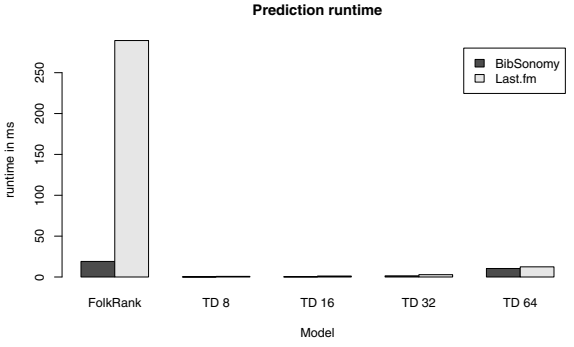


Fig. 7.6 Runtime comparison for predicting one ranked list of tags for the small BibSonomy and the larger Last.fm dataset. FolkRank is compared to a Tucker decomposition (e.g. RTF, HOSVD) with an increasing number of dimensions. On small datasets FolkRank's runtime is feasible but on larger datasets it gets impractical. In contrast to this, Tucker factorization models only depend on the factorization dimensions and not on the size of the dataset.

Our approach at the ECML/PKDD Challenge (Rendle and Schmidt-Thieme, 2009) had two additions to the BPoR-PITF presented here: (1) In the challenge, the recommender could benefit from suggesting lists with less than 5 tags – thus we estimated how many tags to recommend. Even without this enhancement for the challenge, our approach would still have the best score with 0.345. (2) We ensembled many BPoR-PITF models to reduce variance in the ranking estimates. On our holdout test, this improved the result only a little bit.

7.6.3.2 Runtime

Our results for the prediction quality indicate that BPoR-PITF and RTF-TD outperform other approaches. Next, we want to investigate the runtime for prediction and learning.

Prediction runtime

Fast predictions are crucial for applying tag recommenders. We compare the factorization model with the largest prediction complexity (i.e. Tucker decomposition) to Folkrank which achieves the best quality among the non-factorization approaches.

The empirical runtime comparison for predicting a ranked list of tags for a post can be found in figure 7.6. As you can see, the runtime of a TD model[5] is dominated by the dimension of the factorization and is more or less independent of the size of the dataset. The runtime on the BibSonomy dataset and the 20 times larger Last.fm dataset are almost the same – e.g. for $k = 64$ 10.4 ms for BibSonomy and 12.4 ms

[5] Note that the prediction runtime complexity of a Tucker decomposition model is independent of the optimization (e.g. RTF, BPoR, HOSVD).

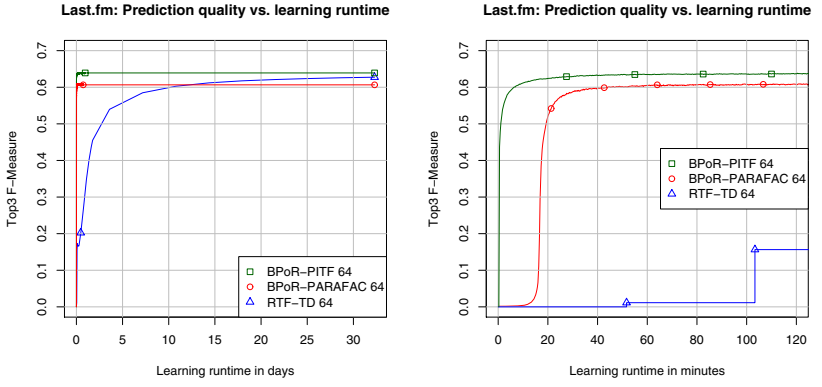


Fig. 7.7 F-Measure on top-3 list after training a model for x days/ hours. Learning a high quality TD model (RTF-TD) on a larger dataset like Last.fm takes several days. The PITF and PARAFAC models give good prediction quality already after 20 and 40 minutes respectively.

for Last.fm. With smaller factorization, the number of tags has a larger influence on the runtime – e.g. for $k = 16$ it is 0.3 ms vs. 1.1 ms. For the very large factorization of $k = 128$ and the very small dataset of BibSonomy, the prediction runtime of TD is worse than that of Folkrank (82.1 ms vs 19.1 ms). The reason is that the runtime of FolkRank depends on the size of the dataset – i.e. the observations S – and on the very small BibSonomy dataset that leads to a reasonable runtime but already for the larger Last.fm dataset the runtime of FolkRank is not feasible any more for real-time predictions. These results match to the theoretical prediction complexity which is $O(\text{iter}(|S|+|U|+|I|+|T|)+|T|N)$ for Folkrank (Jäschke et al, 2008) but only $O(|T| \cdot k_T + k_U \cdot k_I \cdot k_T)$ for TD.

Another major advantage of factorization models is that the tradeoff between quality and speed can be chosen by controlling the number of dimensions. That means depending on the application one can chose if runtime is more important than quality and thus reduce the number of dimensions. With FolkRank this tradeoff cannot be controlled.

The drawback of factorization models in comparison to FolkRank is that they need a training phase. But training is usually done offline and for online updating a factorization model there are very promising results for the related model class of regularized matrix factorization (Rendle and Schmidt-Thieme, 2008). Next, we investigate the training speed of various factorization models.

Learning runtime

The comparison of the convergence of BPoR-PITF to BPoR-PARAFAC and RTF-TD on the Last.fm dataset can be found in figure 7.7. Here you can see how the prediction quality improves after training a model (k=64) for a given time span. The

left chart shows the quality over a span of 30 days. RTF-TD needs about 12 days to achieve a prediction quality as good as BPoR-PARAFAC. Even after 30 days of training, the quality of RTF-TD is still worse than BPoR-PITF.

In contrast to this, BPoR-PITF and BPoR-PARAFAC converge much faster. The right chart shows the quality over the first two hours. BPoR-PITF and BPoR-PARAFAC achieve convergence already after 20 and 40 minutes respectively. As each iteration of RTF-TD takes more than 50 minutes, the progress is very slow. When comparing BPoR-PITF and BPoR-PARAFAC among each other, one can see, that BPoR-PITF converges faster. It is interesting to see that in the beginning BPoR-PARAFAC seems to need several updates (18 minutes) before the quality improves reasonably. One explanation could be that BPoR-PARAFAC is searching the structure among the three-way interactions whereas in BPoR-PITF this is already given by the two pairwise interactions.

The worse empirical runtime results of RTF-TD in comparison to BPoR-PARAFAC and BPoR-PITF match to the theoretical runtime complexity analysis of the model equations (see chapter 5). Furthermore, learning for both BPoR-PARAFAC and BPoR-PITF can be easily parallelized because quadruples of two draws usually share no parameters – in contrast to this, all entries in RTF-TD share the core tensor which makes it more difficult to parallelize RTF-TD.

7.7 Conclusion

In this work, we have applied the context-aware ranking framework to the task of tag recommendation. Therefore, we have derived *Bayesian Post-aware Ranking* from BCR which leads to the optimization criterion BPoR-Opt and the learning algorithm BPoR-Learn. As models we have applied the PITF and PARAFAC models that have linear complexity. Furthermore, we have introduced RTF which optimizes the TD model for post-wise AUC and have shown how to speedup learning for RTF.

We have compared our factorization approaches to the state-of-the-art personalized tag recommenders FolkRank, PageRank and HOSVD as well as to the upper bound for any non-personalized tag recommender. Our results indicate that BPoR-PITF and RTF-TD have the best prediction quality outperforming all other approaches. Furthermore, our factorization methods have a much better prediction runtime which makes them applicable for real-world scenarios. Finally, we have empirically shown that our model PITF largely outperforms RTF-TD in learning runtime and achieves better prediction quality on datasets of large scale. The empirical comparison was done on lab experiments and on the 'ECML/ PKDD Discovery Challenge 2009', that PITF has won.

References

Heymann, P., Ramage, D., Garcia-Molina, H.: Social tag prediction. In: SIGIR 2008: Proceedings of the 31st annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 531–538. ACM, New York (2008)

Hotho, A., Jäschke, R., Schmitz, C., Stumme, G.: Information retrieval in folksonomies: Search and ranking. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 411–426. Springer, Heidelberg (2006)

Jäschke, R., Marinho, L., Hotho, A., Schmidt-Thieme, L., Stumme, G.: Tag recommendations in folksonomies. In: Proceedings of the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), Warsaw, Poland (2007)

Jäschke, R., Marinho, L., Hotho, A., Schmidt-Thieme, L., Stumme, G.: Tag recommendations in social bookmarking systems. AICOM (2008)

Ju, S., Hwang, K.B.: A weighting scheme for tag recommendation in social bookmarking systems. In: Proceedings of the ECML-PKDD Discovery Challenge Workshop (2009)

Lathauwer, L.D., Moor, B.D., Vandewalle, J.: A multilinear singular value decomposition. SIAM J. Matrix Anal. Appl. 21(4), 1253–1278 (2000)

Lathauwer, L.D., Moor, B.D., Vandewalle, J.: On the best rank-1 and rank-(r1,r2,.,rn) approximation of higher-order tensors. SIAM J. Matrix Anal. Appl. 21(4), 1324–1342 (2000)

Lipczak, M., Hu, Y., Kollet, Y., Milios, E.: Tag sources for recommendation in collaborative tagging systems. In: Proceedings of the ECML-PKDD Discovery Challenge Workshop (2009)

Marinho, L.B., Preisach, C., Schmidt-Thieme, L.: Relational classification for personalized tag recommendation. In: Proceedings of the ECML-PKDD Discovery Challenge Workshop (2009)

Rendle, S., Schmidt-Thieme, L.: Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In: RecSys 2008: Proceedings of the 2008 ACM Conference on Recommender Systems, pp. 251–258. ACM, New York (2008)

Rendle, S., Schmidt-Thieme, L.: Factor models for tag recommendation in bibsonomy. In: Proceedings of the ECML-PKDD Discovery Challenge Workshop (2009)

Rendle, S., Marinho, L.B., Nanopoulos, A., Schmidt-Thieme, L.: Learning optimal ranking with tensor factorization for tag recommendation. In: KDD 2009: Proceeding of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, New York (2009)

Rennie, J.D.M., Srebro, N.: Fast maximum margin matrix factorization for collaborative prediction. In: ICML 2005: Proceedings of the 22nd International Conference on Machine learning, pp. 713–719. ACM, New York (2005)

Song, Y., Zhang, L., Giles, C.L.: A sparse gaussian processes classification framework for fast tag suggestions. In: CIKM 2008: Proceeding of the 17th ACM Conference on Information and knowledge Management, pp. 93–102. ACM, New York (2008)

Song, Y., Zhuang, Z., Li, H., Zhao, Q., Li, J., Lee, W.C., Giles, C.L.: Real-time automatic tag recommendation. In: SIGIR 2008: Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 515–522. ACM, New York (2008)

Symeonidis, P., Nanopoulos, A., Manolopoulos, Y.: Tag recommendations based on tensor dimensionality reduction. In: RecSys 2008: Proceedings of the 2008 ACM Conference on Recommender Systems, pp. 43–50. ACM, New York (2008)

Tucker, L.: Some mathematical notes on three-mode factor analysis. Psychometrika 31, 279–311 (1966)

Wetzker, R., Said1, A., Zimmermann, C.: Understanding the user: Personomy translation for tag recommendation. In: Proceedings of the ECML-PKDD Discovery Challenge Workshop (2009)

Wetzker, R., Zimmermann, C., Bauckhage, C., Albayrak, S.: I tag, you tag: translating tags for advanced user models. In: WSDM 2010: Proceedings of the third ACM International Conference on Web search and Data Mining, pp. 71–80. ACM, New York (2010)

Zhang, N., Zhang, Y., Tang, J.: A tag recommendation system based on graph. In: Proceedings of the ECML-PKDD Discovery Challenge Workshop (2009)