Learning Optimal Ranking with Tensor Factorization for Tag Recommendation

Steffen Rendle, Leandro Balby Marinho,
Alexandros Nanopoulos, Lars Schmidt-Thieme
Information Systems and Machine Learning Lab (ISMLL)
Institute for Computer Science
University of Hildesheim, Germany
{srendle,marinho,nanopoulos,schmidt-thieme}@ismll.uni-hildesheim.de

ABSTRACT

Tag recommendation is the task of predicting a personalized list of tags for a user given an item. This is important for many websites with tagging capabilities like last.fm or delicious. In this paper, we propose a method for tag recommendation based on tensor factorization (TF). In contrast to other TF methods like higher order singular value decomposition (HOSVD), our method RTF ('ranking with tensor factorization') directly optimizes the factorization model for the best personalized ranking. RTF handles missing values and learns from pairwise ranking constraints. Our optimization criterion for TF is motivated by a detailed analysis of the problem and of interpretation schemes for the observed data in tagging systems. In all, RTF directly optimizes for the actual problem using a correct interpretation of the data. We provide a gradient descent algorithm to solve our optimization problem. We also provide an improved learning and prediction method with runtime complexity analysis for RTF. The prediction runtime of RTF is independent of the number of observations and only depends on the factorization dimensions. Besides the theoretical analysis, we empirically show that our method outperforms other state-of-the-art tag recommendation methods like FolkRank, PageRank and HOSVD both in quality and prediction runtime.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*Parameter learning*

General Terms

Algorithms, Experimentation, Measurement, Performance

Keywords

Tensor factorization, ranking, tag recommendation

1. INTRODUCTION

Tagging, in general, allows users to describe an item (e.g. website, song, friend, ...) with a list of words ('tags'). Tags can be used e.g. for organizing, browsing and searching. Tagging is a popular feature of many websites like last.fm, delicious, facebook, flickr[1]. With tag recommendation a website can simplify the tagging process for a user by recommending tags that the user might want to give for an item. As different users tend to give different tags for the same item, it is important to personalize the recommended tags for an individual user. That means the tag recommender should infer from the already given tags, which tags a certain user is likely to give for a specific item. For predicting a personalized list of tags for an item, the tag recommender should use the tagging behaviour of the past of this and other users as well as the tags for this and other items. Interesting about tagging data is that it forms a ternary relation between users, items and tags. This makes it different from typical recommender systems where the relation is usually binary between users and items. Exploiting all information of the ternary relation is a key challenge in tag recommendation. A second major challenge for tag recommendation is the data interpretation as usually only positive feedback is present in a tagging system.

In this paper we present a tag recommender that is based on a tensor factorization (TF) model and thus can exploit directly the ternary relationship in tagging data [14]. We will show that other learning methods for tensor factorization proposed by now – like HOSVD [7] or other least square methods [8] – are not optimal for learning a TF model for tag recommendation. We will discuss this in detail and propose a new optimization criterion and learning algorithm that directly optimizes a TF model for optimal ranking.

In all our contributions are as follows:

1. We present a new interpretation scheme for tagging data that is able to handle missing values and only poses ranking constraints. This leads to a more accurate interpretation than the typically used '0/1 scheme'.
2. We propose RTF, an optimization criterion and learning algorithm for TF models, that uses our new data interpretation scheme and optimizes the factorization for optimal ranking.
3. Finally, we show empirically that our proposed method

[1]`http://www.last.fm/`, `http://delicious.com/`, `http://www.facebook.com/` and `http://www.flickr.com/`

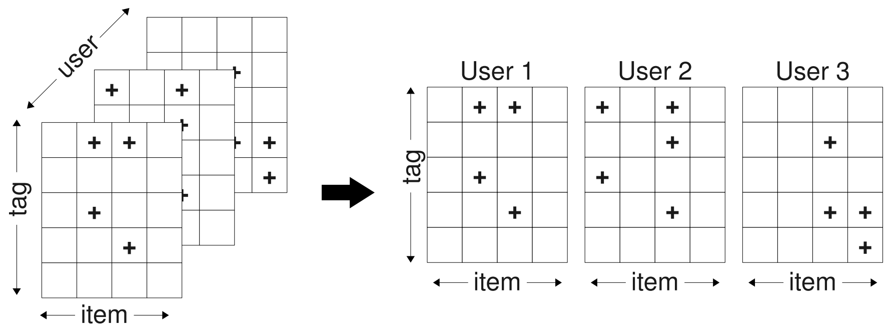

Figure 1: The observed positive examples (u, i, t) are a ternary relationship that can be seen as a 3 dimensional tensor (cube). For each user a matrix is given that contains the tags given for a specific item.

RTF outperforms the best personalized tag recommendation algorithms both in quality and prediction runtime.

2. RELATED WORK

Personalized Tag Recommenders. The literature concerning the problem of personalized tag recommendation is still young, but has nevertheless attracted significant attention recently. In [5] a comprehensive evaluation and comparison of several state-of-the-art tag recommendation algorithms in three different real world datasets is provided. The best results reported in terms of precision and recall, were given by the FolkRank algorithm [3], an adaptation of the well known PageRank. 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. We will show that our method RTF outperforms FolkRank both in quality and prediction runtime.

Non-personalized Tag Recommenders. 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. [2, 13, 12]. In [13], for example, an algorithm based on a Poisson Mixture Model is introduced. 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 and the work presented before, is that their method is content aware. In [12] the problem of tag recommendations is casted as a multilabel 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 5.4.3) indicates that if user information is present, our proposed personalized tag recommender outperforms any non-personalized tag recommender.

Tensor Factorization. While the idea of computing low rank approximations for tensors has already been used for many purposes [7, 11, 6], it has just recently been applied for the problem of personalized tag recommendations [14]. In this approach, HOSVD [7] is applied for computing a low rank approximation of the original tensor, through which tag recommendations are generated yielding promising results. Nevertheless all these TF approaches like HOSVD or other least-square methods [8] do not lead to optimal factorizations for the task of tag recommendation as we will show in this paper both theoretically and empirically.

3. TAG RECOMMENDATION

The task of tag recommendation is to provide a user with a personalized ranked list of tags for a specific item. An example is a bookmark website where after the user has added a new bookmark, the system recommends him a personalized list of ranked tags/ keywords for this bookmark. The list of recommended bookmarks can be learned from the tagging behaviour of the past of this user for other bookmarks and the tagging behaviour of other users for both this and other bookmarks.

3.1 Formalization

Let U be the set of all users, I the set of all items/ resources and T the set of all tags. The tagging information of the past, i.e. all individual tags the users have given to resources, is denoted by $S \subseteq U \times I \times T$. E.g. $(u, i, t) \in S$ would mean that user u has tagged an item i with the tag t. The ternary relation S can be viewed as a three dimensional tag cube (see figure 1), where the dimensions are the users, items and tags. The *posts* P_S denotes the set of all distinct user/ item combinations in S:

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

From the ternary relation S one can induce a tensor Y with training data. There are several ways how to interpret S and create Y. We will present two methods in section 3.2.

The task of tag recommendation is to predict which tags a user u is most likely to use for tagging an item i. That means a tag recommender has to predict the numerical values $\hat{y}_{u,i,t}$ of the tensor $\hat{Y}$ indicating how much the user likes a tag for an item. Instead of predicting single elements of $\hat{Y}$, in general the system should provide the user a personalized list of the best N tags for the item. Given a predictor $\hat{Y}$ the list Top of the N highest scoring items for a given user u and an item i can be calculated by:

$$\text{Top}(u, i, N) := \underset{t \in T}{\overset{N}{\text{argmax}}}\, \hat{y}_{u,i,t} \tag{1}$$

Where the superscript N denotes the number of tags to return.

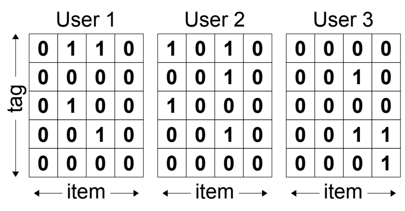

Figure 2: 0/1 interpretation: Positive examples are encoded as 1 and the rest as 0.

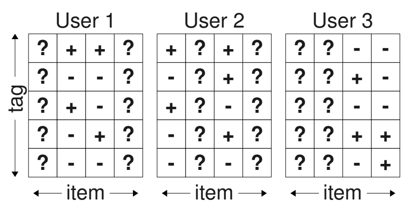

Figure 3: Post-based ranking interpretation: Non observed data inside given posts are negative examples. All other entries are missing values. No numeric value is assigned to the classes, instead only a ranking is implied.

3.2 Interpretation of the Data

For any learning algorithm good training data is crucial. In typical learning tasks, the set of positive and negative examples is clearly given. In contrast to this in many recommender problems, like in tag recommendation, only positive examples are present. In tag recommendation the positive examples are the elements of S. But it is unclear how the rest of this relation $(U \times I \times T) \setminus S$ should be interpreted.

3.2.1 *0/1 Interpretation scheme*

A common interpretation scheme – we call it the *0/1 scheme* – is to encode positive feedback as 1 and interprete the remaining data as 0 (see figure 2). The training data $Y^{0/1}$ is then defined as:

$$y^{0/1}_{u,i,t} = \begin{cases} 1, & (u,i,t) \in S \\ 0, & \text{else} \end{cases}$$

This interpretation is e.g. used for training tag recommenders using a HOSVD model [14].

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. 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 5.1), the 0 values dominate the 1 by many orders of magnitude. To give a practical example, first the sparsity for 0/1 interpretation is:

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

   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.

3.2.2 *Post-based Ranking Interpretation Scheme*

In this paper we present another interpretation scheme, that we call the *post-based ranking interpretation*. Our scheme addresses all of the three problems of the '0/1 scheme'. With this interpretation we distinguish between positive and negative examples and missing values. The idea is that positive and negative examples are only generated from observed posts. All other entries – e.g. all tags for an item that a user has not tagged yet – are assumed to be missing values (see figure 3). First we define the set of positive and negative examples for a given post:

$$T^+_{u,i} := \{t \,|\, (u,i) \in P_S \wedge (u,i,t) \in S\}$$
$$T^-_{u,i} := \{t \,|\, (u,i) \in P_S \wedge (u,i,t) \notin S\}$$

From this we can define for the values of Y pairwise ranking constraints:

$$y^p_{u,i,t_1} > y^p_{u,i,t_2} \Leftrightarrow (u,i,t_1) \in T^+_{u,i} \wedge (u,i,t_2) \in T^-_{u,i} \qquad (2)$$

From a semantical point of view this scheme makes more sense as the user/ item combinations that have no tags are the ones that the recommender system will have to predict 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'[2]. Also inside a given post the negative values 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. 2). In all, a model for 'post-based ranking interpretation' should be optimized to satisfy as many ranking constraints as possible. Please note that optimizing for the ranking constraints between positive and negative values is related to optimizing the ranking statistic AUC (area under the ROC-curve) as we will see in the next section.

4. RANKING WITH TENSOR FACTORIZATION (RTF)

First we describe tensor factorization models in general. Then we present in detail how a tensor factorization model can be learned for optimizing the ranking statistic AUC (area under the ROC-curve). We discuss the RTF model and compare it to HOSVD and Folkrank.

[2]Please note that the '0/1 scheme' poses more constraints on $y^p_{u,i,t}$ as fitting to 0/1 is required and there are constraints on tags of non-observed posts.

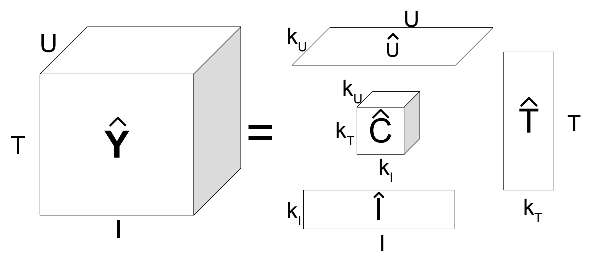

Figure 4: Tensor factorization: The tensor $\hat{Y}$ is constructed by multiplying three features matrices $\hat{U}$, $\hat{I}$ and $\hat{T}$ to a small core tensor $\hat{C}$.

4.1 Tensor Factorization Model

With tensor factorization, Y is estimated by three low rank matrices and one tensor (see figure 4). For each of the three dimensions – i.e. user, items and tags – one of the low rank matrices tries to represent an entity with a small number of parameters. We call the matrices *feature matrices* and the tensor *core tensor*. The model parameters of a TF model can be seen as latent variables.

The prediction is made by multiplying the three feature matrices to the core tensor:

$$\hat{Y} := \hat{C} \times_u \hat{U} \times_i \hat{I} \times_t \hat{T} \tag{3}$$

Where the core tensor $\hat{C}$ and the feature matrices $\hat{U}$, $\hat{I}$ and $\hat{T}$ are the model parameters that have to be learned and $\times_x$ is the tensor product to multiply a matrix on dimension x with a tensor. The model parameters have the following sizes:

$$\hat{C} \in \mathbb{R}^{k_U \times k_I \times k_T}, \quad \hat{U} \in \mathbb{R}^{|U| \times k_U}$$
$$\hat{I} \in \mathbb{R}^{|I| \times k_I}, \quad \hat{T} \in \mathbb{R}^{|T| \times k_T}$$

Where k_U, k_I and k_T are the dimensions of the low-rank approximation. That means that $\hat{Y}$ in the formula (3) results in a tensor with dimensions $|U| \times |I| \times |T|$. We denote the model parameters by the quadruple $\hat{\theta} := (\hat{C}, \hat{U}, \hat{I}, \hat{T})$.

Given the feature matrices and the core tensor, the prediction $\hat{y}_{u,i,t}$ can be made as follows:

$$\hat{y}_{u,i,t} = \sum_{\tilde{u}} \sum_{\tilde{i}} \sum_{\tilde{t}} \hat{c}_{\tilde{u},\tilde{i},\tilde{t}} \cdot \hat{u}_{u,\tilde{u}} \cdot \hat{i}_{i,\tilde{i}} \cdot \hat{t}_{t,\tilde{t}} \tag{4}$$

Given $\hat{y}_{u,i,t}$ a personalized ranked list of tags for user u and item i can be created with formula (1).

Throughout the paper, indices over the feature dimension of a feature matrix are marked with a tilde (e.g. $\tilde{t}$) and elements of a feature matrix are marked with a hat (e.g. $\hat{t}_{t,\tilde{t}}$).

4.2 Learning to Rank with Tensor Factorization

After we have presented the model equation (4), we now show how to learn the model parameters $\hat{C}$, $\hat{U}$, $\hat{I}$ and $\hat{T}$. First we discuss the optimization criterion and afterwards we derive an algorithm for the optimization task.

4.2.1 *Optimization Criterion*

For finding the 'best' model parameters an optimization criterion has to be defined. Usually tensor factorization models (like HOSVD) are learned by minimizing an element-wise loss on the elements of $\hat{Y}$ – e.g. by optimizing the square loss:

$$\operatorname*{argmin}_{\hat{\theta}} \sum_{(u,i,t) \in U \times I \times T} (\hat{y}_{u,i,t} - y_{u,i,t})^2$$

For this minimization task, one can use standard HOSVD or square-loss implementations like [6, 8], because the data Y is assumed to be dense. Such an optimization uses the '0/1 interpretation scheme' (see section 3.2). As we have argued before this scheme misinterprets the semantics of the data as it does not handle missing values, suffers from sparsity in terms of domination of zero values and does not optimize for ranking quality.

Instead we propose another optimization criterion that uses the 'post-based ranking interpretation' and maximizes the ranking statistic AUC (area under the ROC-curve). The quality measure AUC (or Mann-Whitney statistic) for a given post of user u for item i is defined as:

$$\mathrm{AUC}(\hat{\theta}, u, i) := \frac{1}{|T^+_{u,i}||T^-_{u,i}|} \sum_{t^+ \in T^+_{u,i}} \sum_{t^- \in T^-_{u,i}} H_{0.5}(\hat{y}_{u,i,t^+} - \hat{y}_{u,i,t^-}) \tag{5}$$

where H_α is the Heaviside function:

$$H_\alpha := \begin{cases} 0, & x < 0 \\ \alpha, & x = 0 \\ 1, & x > 0 \end{cases} \tag{6}$$

The overall optimization task with respect to the ranking statistic AUC and the observed data is then:

$$\operatorname*{argmax}_{\hat{\theta}} \sum_{(u,i) \in P_S} \mathrm{AUC}(\hat{\theta}, u, i) \tag{7}$$

With this optimization (i) missing values are taken into account because the maximization is only done on the observed posts P_S and (ii) the model is optimized for ranking. In all, this criterion takes into account all obeservations of section 3.2.

4.2.2 *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 [10]. Adding a regularization objective to the optimization task in formula (7) leads to the following objective:

$$\operatorname*{argmax}_{\hat{\theta}} \sum_{(u,i) \in P_S} \mathrm{AUC}(\hat{\theta}, u, i) - \left(\gamma_C ||\hat{C}||_F^2 + \gamma ||\hat{U}||_F^2 + \gamma ||\hat{I}||_F^2 + \gamma ||\hat{T}||_F^2\right) \tag{8}$$

Where γ_C and γ are the regularization parameters for the core tensor and the feature matrices respectively. $|| \cdot ||_F^2$ is the Frobenius norm.

4.2.3 *Learning Algorithm*

Next we present an algorithm to solve the optimization problem of formula (8). Obviously, optimizing (8) directly is infeasible. Instead we use gradient descent to minimize the objective function. As the AUC is not differentiable because of the Heaviside function, we replace H like in [1] by the s-shaped logistic function s:

$$s(x) := \frac{1}{1+e^{-x}}$$

The overall algorithm can be found in figure 5. This algorithm uses a stochastic update approach, that means for each post $(u,i) \in P_S$ the model parameters are updated.

For using gradient descent, AUC has to be differentiated with respect to all model parameters. First of all, the derivative of AUC given a post $(u,i) \in P_S$ can be simplified for all model parameters x:

$$\begin{aligned}
&\frac{\partial}{\partial x}\,\mathrm{AUC}(\hat\theta,u,i)\\
=&\frac{\partial}{\partial x}\frac{1}{|T^+_{u,i}||T^-_{u,i}|}\sum_{t^+\in T^+_{u,i}}\sum_{t^-\in T^-_{u,i}} s(\hat y_{u,i,t^+}-\hat y_{u,i,t^-})\\
=&z\sum_{t^+\in T^+_{u,i}}\sum_{t^-\in T^-_{u,i}} w_{t^+,t^-}\frac{\partial}{\partial x}(\hat y_{u,i,t^+}-\hat y_{u,i,t^-})
\end{aligned}$$

with:

$$\begin{aligned}
w_{t^+,t^-} &:= s(\hat y_{u,i,t^+}-\hat y_{u,i,t^-})(1-s(\hat y_{u,i,t^+}-\hat y_{u,i,t^-}))\\
z &:= \frac{1}{|T^+_{u,i}||T^-_{u,i}|}\\
\hat y_{u,i,t^+}-\hat y_{u,i,t^-} &= \sum_{\tilde u}\sum_{\tilde i}\sum_{\tilde t}\hat c_{\tilde u,\tilde i,\tilde t}\hat u_{u,\tilde u}\hat i_{i,\tilde i}(\hat t_{t^+,\tilde t}-\hat t_{t^-,\tilde t})
\end{aligned}$$

Hence, the derivative of the core tensor features is:

$$\frac{\partial\,\mathrm{AUC}}{\partial \hat c_{\tilde u,\tilde i,\tilde t}} = z\sum_{t^+\in T^+_{u,i}}\sum_{t^-\in T^-_{u,i}} w_{t^+,t^-}\hat u_{u,\tilde u}\hat i_{i,\tilde i}(\hat t_{t^+,\tilde t}-\hat t_{t^-,\tilde t})$$

For the feature matrices U and I the derivatives are as follows:

$$\frac{\partial\,\mathrm{AUC}}{\partial \hat u_{u,\tilde u}} = z\sum_{t^+\in T^+_{u,i}}\sum_{t^-\in T^-_{u,i}}\sum_{\tilde i}\sum_{\tilde t} w_{t^+,t^-}\hat c_{\tilde u,\tilde i,\tilde t}\hat i_{i,\tilde i}(\hat t_{t^+,\tilde t}-\hat t_{t^-,\tilde t})$$

$$\frac{\partial\,\mathrm{AUC}}{\partial \hat i_{i,\tilde i}} = z\sum_{t^+\in T^+_{u,i}}\sum_{t^-\in T^-_{u,i}}\sum_{\tilde u}\sum_{\tilde t} w_{t^+,t^-}\hat c_{\tilde u,\tilde i,\tilde t}\hat u_{u,\tilde u}(\hat t_{t^+,\tilde t}-\hat t_{t^-,\tilde t})$$

For the tags the updates depend on whether a tag t is positive or negative:

$$\frac{\partial\,\mathrm{AUC}}{\partial \hat t_{t^+,\tilde t}} = -z\sum_{t^-\in T^-_{u,i}}\sum_{\tilde u}\sum_{\tilde i} w_{t^+,t^-}\hat u_{u,\tilde u}\hat i_{i,\tilde i}\hat c_{\tilde u,\tilde i,\tilde t}$$

$$\frac{\partial\,\mathrm{AUC}}{\partial \hat t_{t^-,\tilde t}} = z\sum_{t^+\in T^+_{u,i}}\sum_{\tilde u}\sum_{\tilde i} w_{t^+,t^-}\hat u_{u,\tilde u}\hat i_{i,\tilde i}\hat c_{\tilde u,\tilde i,\tilde t}$$

4.3 Relations to HOSVD

Higher order singular value decomposition (HOSVD) [7] is another method for learning a tensor factorization model. HOSVD targets to create an optimal reconstruction of a tensor Y using the model equation (3). Even though HOSVD is a good method for the task of reconstruction tensors, for the task of personalized ranking HOSVD has three major drawbacks to RTF:

```
procedure LEARNRTF($S, \alpha, \gamma, \gamma_C$)
    initialize $\hat\theta := (\hat C, \hat U, \hat I, \hat T)$
    repeat
        for $(u,i) \in P_S$ do
            for $(\tilde u, \tilde i, \tilde t) \in k_u \times k_i \times k_t$ do
                $\hat c_{\tilde u,\tilde i,\tilde t} \leftarrow \hat c_{\tilde u,\tilde i,\tilde t} + \alpha\left(\frac{\partial AUC}{\partial \hat c_{\tilde u,\tilde i,\tilde t}} - \gamma_C \cdot \hat c_{\tilde u,\tilde i,\tilde t}\right)$
            end for
            for $\tilde u \leftarrow 1,\dots,k_u$ do
                $\hat u_{u,\tilde u} \leftarrow \hat u_{u,\tilde u} + \alpha\left(\frac{\partial AUC}{\partial \hat u_{u,\tilde u}} - \gamma \cdot \hat u_{u,\tilde u}\right)$
            end for
            for $\tilde i \leftarrow 1,\dots,k_i$ do
                $\hat i_{i,\tilde i} \leftarrow \hat i_{i,\tilde i} + \alpha\left(\frac{\partial AUC}{\partial \hat i_{i,\tilde i}} - \gamma \cdot \hat i_{i,\tilde i}\right)$
            end for
            for $t \leftarrow 1,\dots,|T|$ do
                for $\tilde t \leftarrow 1,\dots,k_t$ do
                    $\hat t_{t,\tilde t} \leftarrow \hat t_{t,\tilde t} + \alpha\left(\frac{\partial AUC}{\partial \hat t_{t,\tilde t}} - \gamma \cdot \hat t_{t,\tilde t}\right)$
                end for
            end for
        end for
    until stopping criterion met
    return $\hat\theta$
end procedure
```

Figure 5: Learning RTF models by gradient descent with learning rate α and regularization γ and γ_C.

1. HOSVD cannot deal with missing values. For tag recommendation the missing values are usually filled with zeros [14].
2. HOSVD optimizes for minimal element-wise error. But for the ranking problem of tag recommendation we are interested in another objective function.
3. HOSVD has no regularization. For machine learning tasks preventing overfitting is very important so HOSVD is prone to overfitting.

There are also other related tensor factorization methods similar to HOSVD like iterative least-square error minimization [8], that also suffer from the same problems discussed above. In all HOSVD for tag recommendation tries to optimize the '0/1 interpretation scheme' (see section 3.2). Besides this theoretical analysis, in our evaluation we will show that RTF largely outperforms HOSVD.

4.4 Fast Computation of RTF

One of the benefits from a factorization model like RTF or HOSVD is that after a model is built, predictions only rely on the model. This leads to faster prediction runtime than with models like FolkRank. In the following, we look into the runtime of RTF models in detail and show how to speed them up.

4.4.1 *Prediction*

Formula (1) shows the general way to predict a top-n list of tags for a specific user u and item i. For predicting $\hat y_{u,i,t}$

Method	Prediction	Training
RTF	$O(\|T\| \cdot k_T + k_U \cdot k_I \cdot k_T)$	$O(iter \cdot \|P_S\| \cdot (k_T \cdot \|T\|^2 + k_U \cdot k_I \cdot k_T))$
Folkrank	$O(iter \cdot (\|S\| + \|U\| + \|I\| + \|T\|) + \|T\| \cdot N)$	$O(1)$

Figure 6: Runtime complexity: The prediction runtime of RTF only depends on the small factorization dimensions (k_U, k_I, k_T) and the number of tags T but is independent from the size of observed data S, number of users U and items I.

with a factorization model, formula (4) is used. The runtime complexity for equation (4) 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 (see eq. 1) is $O(|T| \cdot k_U \cdot k_I \cdot k_T)$. Though the runtime can be improved largely by reordering the sums in formula (4):

$$\begin{aligned}
\hat{y}_{u,i,t} &= \sum_{\tilde{u}} \sum_{\tilde{i}} \sum_{\tilde{t}} \hat{c}_{\tilde{u},\tilde{i},\tilde{t}} \cdot \hat{u}_{u,\tilde{u}} \cdot \hat{i}_{i,\tilde{i}} \cdot \hat{t}_{t,\tilde{t}} \\
&= \sum_{\tilde{t}} \hat{t}_{t,\tilde{t}} \cdot \sum_{\tilde{u}} \sum_{\tilde{i}} \hat{c}_{\tilde{u},\tilde{i},\tilde{t}} \cdot \hat{u}_{u,\tilde{u}} \cdot \hat{i}_{i,\tilde{i}} \\
&= \sum_{\tilde{t}} \hat{t}_{t,\tilde{t}} \cdot \hat{t}^{u,i}_{\tilde{t}} \\
\text{with} \quad \hat{t}^{u,i}_{\tilde{t}} &:= \sum_{\tilde{u}} \sum_{\tilde{i}} \hat{c}_{\tilde{u},\tilde{i},\tilde{t}} \cdot \hat{u}_{u,\tilde{u}} \cdot \hat{i}_{i,\tilde{i}}
\end{aligned}$$

When making a top-n prediction for user u and item i instead of computing (4) for each tag, an intermediate result $\hat{t}^{u,i}_{\tilde{t}}$ 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 a factorization 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.

4.4.2 Learning

The complexity of learning a RTF model (see figure 5) is in $O(iter \cdot |P_S| \cdot |T|^2 \cdot k_T \cdot k_I \cdot k_U))$ where *iter* is the number of iterations. This is because the runtime is dominated by the update of tag features. Also here by rearranging the sums for the gradients and storing intermediate results, one can achieve a better runtime. We suggest to calculate the intermediate vector $v_{\tilde{t}}$ which then can be used in the gradient for $\hat{U}$, $\hat{I}$ and $\hat{C}$:

$$v_{\tilde{t}} := \sum_{t^+ \in T^+_{u,i}} \sum_{t^- \in T^-_{u,i}} w_{t^+,t^-} (\hat{t}_{t^+,\tilde{t}} - \hat{t}_{t^-,\tilde{t}})$$

This vector can be calculated in $O(|T^+| \cdot |T^-| \cdot k_T)$. With this definitions, the gradients can be simplified to:

$$\begin{aligned}
\frac{\partial \,\text{AUC}}{\partial \hat{c}_{\tilde{u},\tilde{i},\tilde{t}}} &= z \cdot \hat{u}_{u,\tilde{u}} \cdot \hat{i}_{i,\tilde{i}} \cdot v_{\tilde{t}} \\
\frac{\partial \,\text{AUC}}{\partial \hat{u}_{u,\tilde{u}}} &= z \sum_{\tilde{i}} \sum_{\tilde{t}} \hat{c}_{\tilde{u},\tilde{i},\tilde{t}} \cdot \hat{i}_{i,\tilde{i}} \cdot v_{\tilde{t}} \\
\frac{\partial \,\text{AUC}}{\partial \hat{i}_{i,\tilde{i}}} &= z \sum_{\tilde{u}} \sum_{\tilde{t}} \hat{c}_{\tilde{u},\tilde{i},\tilde{t}} \cdot \hat{u}_{u,\tilde{u}} \cdot v_{\tilde{t}}
\end{aligned}$$

So in total, the runtime for these updates is in $O(|T^+| \cdot |T^-| \cdot k_T + k_U \cdot k_I \cdot k_T)$.

Similarly, the updates for T can be simplified by calculating the intermediate vector $q_{\tilde{t}}$ in $O(k_T \cdot k_U \cdot k_I)$:

$$q_{\tilde{t}} := \sum_{\tilde{u}} \sum_{\tilde{i}} c_{\tilde{u},\tilde{i},\tilde{t}} \cdot \hat{u}_{u,\tilde{u}} \cdot \hat{i}_{i,\tilde{i}}$$

Now the updates for the tag feature matrices can be calculated in a total runtime of $O(k_T \cdot |T|^2 + k_U \cdot k_I \cdot k_T)$ using the formulas:

$$\begin{aligned}
\frac{\partial \,\text{AUC}}{\partial \hat{t}_{t^+,\tilde{t}}} &= -z \cdot q_{\tilde{t}} \sum_{t^- \in T^-_{u,i}} w_{t^+,t^-} \\
\frac{\partial \,\text{AUC}}{\partial \hat{t}_{t^-,\tilde{t}}} &= z \cdot q_{\tilde{t}} \sum_{t^+ \in T^+_{u,i}} w_{t^+,t^-}
\end{aligned}$$

In all, learning an RTF model can be implemented in $O(iter \cdot |P_S| \cdot (k_T \cdot |T|^2 + k_U \cdot k_I \cdot k_T))$.

4.4.3 Runtime Complexity Comparison

We compare the runtime complexity of our RTF method to the state-of-the-art tag recommendation method FolkRank [4]. Jäschke et al. [5] have proven that the runtime complexity for top-n predictions with FolkRank is $O(iter \cdot (|S| + |U| + |I| + |T|) + |T| \cdot N)$ where *iter* is the number of iterations. That means for predicting a personalized top-n list, FolkRank has to pass several times the whole database of observations. When we compare this to RTF with complexity $O(|T| \cdot k_T + k_U \cdot k_I \cdot k_T)$ it is obvious that RTF models have a much better runtime complexity as they only depend on the small dimensions of the factorization and on the number of tags. The only advantage of FolkRank over RTF w.r.t. runtime is that it does not have a training phase. But as training is usually done offline, this does not affect the applicability of RTF for fast large-scale tag recommendation. In our evaluation chapter we will give an empirical comparison of the prediction runtime of FolkRank and RTF.

5. EVALUATION

We investigate the performance of RTF both in prediction quality and runtime compared to the other state-of-the-art tag recommendation algorithms HOSVD, FolkRank and PageRank.

5.1 Datasets

We evaluate our RTF method on the BibSonomy and Last.fm dataset from [4]. As in [4, 5, 14] we use a p-core[3] – for BibSonomy the 5-core and for last.fm the 10-core. The dataset characteristics of the p-cores are:

dataset	$\|U\|$	$\|I\|$	$\|T\|$	$\|S\|$	$\|P_S\|$
BibSonomy	116	361	412	10,148	2,522
Last.fm	2,917	1,853	2,045	219,702	75,565

[3]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 in at least p posts.

5.2 Evaluation Methodology

We use the common evaluation protocol for tag-recommender of predicting posts [5]. For each user in the dataset we remove all triples S_{test} he has given for one item – i.e. we remove one post for each user. The remaining observed user-item-tag triples are the training set $S_{train} := S \setminus S_{test}$. Then we learn the models on S_{train} and predict top-N lists for each of the removed posts $P_{S_{test}}$. We measure the recall and precision of the top-1, top-2 to top-10 lists of each post and report for each top-N level (1 to 10) the F1-measures of the average recall and precision:

$$\mathrm{Prec}(S_{test}, N) := \underset{(u,i)\in P_{S_{test}}}{\mathrm{avg}} \frac{|\,\mathrm{Top}(u,i,N) \cap \{t|(u,i,t) \in S_{test}\}|}{N}$$

$$\mathrm{Recall}(S_{test}, N) := \underset{(u,i)\in P_{S_{test}}}{\mathrm{avg}} \frac{|\,\mathrm{Top}(u,i,N) \cap \{t|(u,i,t) \in S_{test}\}|}{|\{t|(u,i,t) \in S_{test}\}|}$$

$$\mathrm{F1}(S_{test}, N) := \frac{2 \cdot \mathrm{Prec}(S_{test}, N) \cdot \mathrm{Recall}(S_{test}, N)}{\mathrm{Prec}(S_{test}, N) + \mathrm{Recall}(S_{test}, N)}$$

We choose F1-Measure on top-n lists as the main quality measure so that the results can be directly compared to related work like [5]. Additionally, we also report the related measure AUC for the RTF models. All experiments are repeated 10 times and we report the mean of the runs. For each run, we use exactly the same train/ test splits as in [5].

We run RTF with $(k_u, k_i, k_t) \in \{(8, 8, 8), (16, 16, 16), (32, 32, 32), (64, 64, 64)\}$ dimensions; for BibSonomy we also run $(k_u, k_i, k_t) = (128, 128, 128)$ dimensions. The corresponding model is called "RTF 8", "RTF 16", and so on. The other 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 $\hat{\theta}$ are initialized with small random values drawn from the normal distribution $N(0, 0.1)$.

For FolkRank and PageRank we report the values obtained by [5] as we use the same datasets and splits. 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. As with FolkRank and PageRank, all hyperparameters were optimized on one split and then used for all the other splits.

For the runtime comparison for prediction we used a C++ implementation of Folkrank and an Object-Pascal implementation of RTF.

5.3 Repeatability of Experiments

Both the datasets and the implementations of all algorithms of our experiments are publicly available for research purposes. The BibSonomy dataset we used is available from the University of Kassel [4]. We will provide our Last.fm dataset upon request by email. FolkRank and PageRank is provided by the University of Kassel within the Nepomuk project[4]. The HOSVD of our experiments [6] is available as Mathlab package[5]. Our RTF implementation is available upon request by email.

[4]http://dev.nepomuk.semanticdesktop.org/download/
[5]http://csmr.ca.sandia.gov/~tgkolda/TensorToolbox/

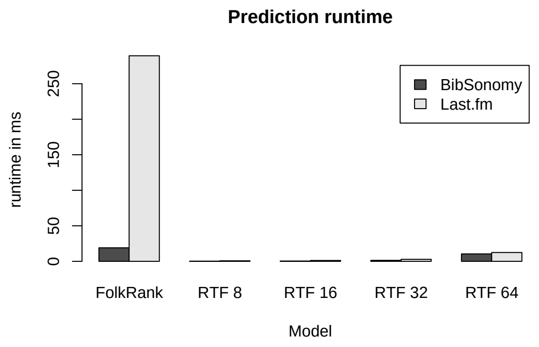

Figure 9: Runtime comparison for predicting one ranked list of tags for the small BibSonomy and the larger Last.fm dataset. FolkRank is compared to RTF with an increasing number of dimensions. On small datasets FolkRank's runtime is feasible but on larger datasets it gets impractical. In contrast to this RTF only depends on the factorization dimensions and not on the size of the dataset.

5.4 Results and Discussion

In the following, we discuss the results of our evaluation. Figure 7 shows a qualitative comparison of the state-of-the-art models FolkRank, HOSVD and PageRank to our model class RTF. There you can see that RTF models with a sufficient number of dimensions (e.g. 64) outperform all other models in quality. In figure 8 you can see the increasing AUC quality of RTF models with an increasing number of dimensions. Finally figure 9 compares the prediction runtime of FolkRank to the runtime of RTF models.

5.4.1 *RTF vs. FolkRank*

When comparing the prediction quality of RTF and FolkRank (figure 7) one can see that high dimensional RTF models outperform FolkRank on both datasets in quality. On BibSonomy RTF with 64/ 128 dimensions achieves comparable results whereas on the larger Last.fm dataset already 32 dimensions clearly outperform FolkRank in quality.

An empirical runtime comparison for predicting a ranked list of tags for a post can be found in figure 9. As you can see, the runtime of the RTF model is dominated by the dimension of the factorization and is 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 RTF64 10.4 ms for BibSonomy and 12.4 ms for Last.fm. With smaller factorization, the number of tags has a larger influence on the runtime – e.g. for RTF16 it is 0.3 ms vs. 1.1 ms. For the very large factorization of RTF128 and the very small dataset of BibSonomy, the runtime of RTF 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. All these empirical results match the theoretical complexity analysis we presented in section 4.4.3.

Another major advantage of RTF is that the tradeoff be-

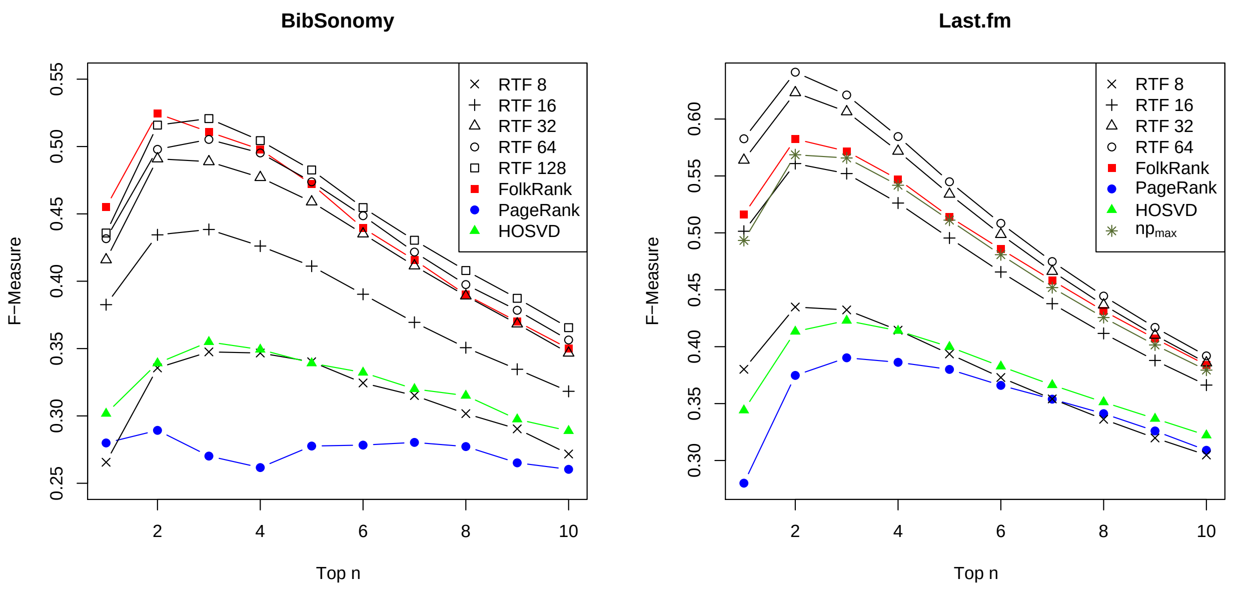

Figure 7: F-Scores for Top-1, Top-2 to Top-10 lists on two datasets. The best FolkRank, PageRank and HOSVD results are compared to RTF with an increasing number of dimensions. – For Last.fm np$_{\text{max}}$ is the theoretical upper (!) bound for non-personalized tag recommenders (see section 5.4.3).

tween 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 you cannot control this tradeoff.

The only drawback of RTF to FolkRank is that it needs 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 [9].

5.4.2 RTF vs. HOSVD

The prediction quality of RTF is clearly superior to the one of HOSVD (figure 7). On BibSonomy even with a very small number of 8 dimensions, RTF achieves almost similar results as HOSVD with a dimensionality of $(60, 105, 225)$ and $(875, 556, 614)$ respectivly. 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. The empirical qualitative results match our discussion about the data interpretation in section 3.2.

Even though RTF and HOSVD have the same prediction method and thus prediction complexity, in practice RTF models are much faster in prediction than comparable HOSVD models, because RTF 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 we can choose the same number of dimensions for user, item and tags. Furthermore for RTF, by increasing the number of dimensions we get better results. We expect this behaviour due to the regularization of RTF models.

5.4.3 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 recommender np$_{\text{max}}$ (see figure 7). The weighting method for np$_{\text{max}}$ is:

$$\hat{y}^{\text{np}_{\max}}_{u,i,t} := |\{u^* | (u^*, i, t) \in S_{test}\}|$$

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 [2, 13, 12] is guaranteed to have a lower (or in the best case the same) quality on S_{test}. As you can see in figure 7, personalized tag recommenders like FolkRank, RTF32 and RTF64 outperform np$_{\text{max}}$ the theoretical upper bound for non-personalized tag recommendation[6]. That means, in applications, where there is personalized information present, personalized tag recommender are supposed to outperform non-personalized tag recommender.

6. CONCLUSION AND FUTURE WORK

In this paper, we have presented a new optimization criterion for tensor factorization for the task of ranking with missing values. Our optimization is motivated theoretically by a proper interpretation of observed and non-observed

[6]Evaluating np$_{\text{max}}$ 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.

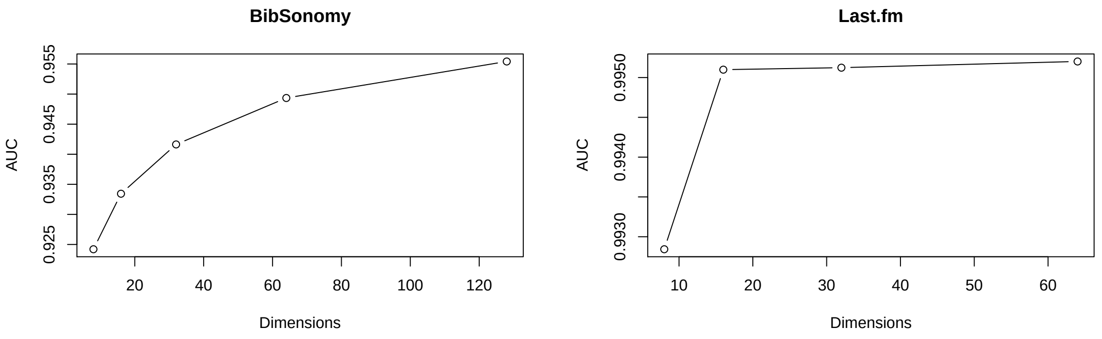

Figure 8: Area under the ROC-curve values for ranking with RTF. The dimensions of the RTF model are increased from 8 to 128 and 64 respectively.

data. It can handle both missing values and pairwise ranking constraints. In all, it is focused on learning the best ranking instead of optimizing for minimal element-wise error like in other tensor factorization algorithms (e.g. HOSVD). For our proposed optimization task, we have presented an optimization algorithm based on gradient descent. In our evaluation we have shown that this algorithm largely outperforms HOSVD in quality – even with much less factorization dimensions which leads to higher prediction speed than HOSVD. Furthermore we have shown that our method is also able to outperform other state-of-the-art tag recommendation algorithms like FolkRank and PageRank in quality and largely in prediction runtime.

In future work, we want to study the isolated effect of each single improvement over HOSVD, namely data interpretation, regularization and AUC optimization.

Acknowledgments

This work is partially co-funded through the European Commission FP7 project MyMedia (www.mymediaproject.org) under the grant agreement no. 215006. Leandro Balby Marinho is supported by the Brazilian National Council for Scientific and Technological Research (CNPq).

7. REFERENCES

[1] A. Herschtal and B. Raskutti. Optimising area under the roc curve using gradient descent. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004.

[2] P. Heymann, D. Ramage, and H. Garcia-Molina. Social tag prediction. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 531–538. ACM, 2008.

[3] A. Hotho, R. Jäschke, C. Schmitz, and G. Stumme. *Information Retrieval in Folksonomies: Search and Ranking*. 2006.

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

[5] R. Jäschke, L. Marinho, A. Hotho, L. Schmidt-Thieme, and G. Stumme. Tag recommendations in social bookmarking systems. *AI Communications*, pages 231–247, 2008.

[6] T. G. Kolda and J. Sun. Scalable tensor decompositions for multi-aspect data mining. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008)*, 2008.

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

[8] L. D. Lathauwer, B. D. Moor, and J. Vandewalle. 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.

[9] S. Rendle and L. Schmidt-Thieme. Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems*. ACM, 2008.

[10] J. D. M. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*. ACM, 2005.

[11] A. Shashua and T. Hazan. Non-negative tensor factorization with applications to statistics and computer vision. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 792–799. ACM, 2005.

[12] Y. Song, L. Zhang, and C. L. Giles. A sparse gaussian processes classification framework for fast tag suggestions. In *CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge management*, pages 93–102. ACM, 2008.

[13] Y. Song, Z. Zhuang, H. Li, Q. Zhao, J. Li, W.-C. Lee, and C. L. Giles. Real-time automatic tag recommendation. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 515–522. ACM, 2008.

[14] P. Symeonidis, A. Nanopoulos, and Y. Manolopoulos. Tag recommendations based on tensor dimensionality reduction. In *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems*, pages 43–50. ACM, 2008.