Probabilistic Polyadic Factorization and Its Application to Personalized Recommendation

Yun Chi Shenghuo Zhu Yihong Gong
NEC Laboratories America
10080 N. Wolfe Rd, SW3-350
Cupertino, CA, USA
{ychi,zsh,ygong}@sv.nec-labs.com

Yi Zhang
School of Engineering
University of California Santa Cruz
Santa Cruz, CA, USA
yiz@soe.ucsc.edu

ABSTRACT

Multiple-dimensional, i.e., polyadic, data exist in many applications, such as personalized recommendation and multiple-dimensional data summarization. Analyzing all the dimensions of polyadic data in a principled way is a challenging research problem. Most existing methods separately analyze the marginal relationships among pairwise dimensions and then combine the results afterwards. Motivated by the fact that various dimensions of polyadic data jointly affect each other, we propose a probabilistic polyadic factorization approach to directly model all the dimensions simultaneously in a unified framework. We then show the connection between the probabilistic polyadic factorization and a non-negative version of the Tucker tensor factorization. We provide detailed theoretical analysis of the new modeling framework, discuss implementation techniques for our models, and propose several extensions to the basic framework. We then apply the proposed models to the application of personalized recommendation. Extensive experiments on a social bookmarking dataset, *Delicious*, and a paper citation dataset, *CiteSeer*, demonstrate the effectiveness of the proposed models.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data mining*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Information filtering*; J.4 [**Computer Applications**]: Social and Behavioral Sciences—*Economics*

General Terms

Algorithms, Experimentation, Measurement, Theory

Keywords

Probabilistic Polyadic Factorization, Multiple-dimensional Data, Non-negative Tensor Factorization, Personalized Recommendation, Social Bookmarking

CIKM'08, October 26–30, 2008, Napa Valley, California, USA.

1. INTRODUCTION

Binary relationship exists in many applications and dyadic data analysis has been studied extensively by various researchers. For example, the well known Latent Semantic Indexing (LSI) focuses on dyadic data consisting of term-document pairs. For another example, singular value decomposition has been used to model dyadic data consisting of user-item pairs in collaborative filtering. However, in many applications, data are polyadic, i.e., they are of multiple (greater than 2) dimensions. Social bookmarking is such an example: in a social bookmarking system such as Del.icio.us, a *user* assigns a set of *tags* to a given *url* (which corresponds to a Web page). Here data records are user-tag-url three-dimensional triples. Paper citation analysis is another example: in a paper corpus such as CiteSeer, an *author*, in an article on a specific *keyword*, cites a *reference*. In this example, a data record is an author-keyword-reference triple. This data can have even more dimensions if more information such as second author, publication venue, and publication year, is available.

To analyze polyadic data, an important and challenging issue is how to combine heterogeneous information from different dimensions. Various approaches have been proposed to tackle this challenge. However, most existing research work adopts a two-step approach: first, only relationships between pairs of dimensions are analyzed; second, the obtained pairwise relationships are combined in certain ways. A main weak point of such a two-step approach is that it considers the relationships among different pairs of dimensions *independently* while in reality, different dimensions of polyadic data affect each other in a *joint* way. There are some recent studies [10, 15, 16], that combine the two steps mentioned above into a single-step process. The key idea of these studies is to use *the same* set of concepts to simultaneously capture all the pairwise relationships among different dimensions. These approaches are more accurate in modeling polyadic data, however they have two major weaknesses. First, they usually use a linear combination to fuse all pairwise relationships among different dimensions. Such a linear combination is somewhat ad hoc, which makes it difficult to find a good intuition behind the coefficients as well as a principled way to set the values of the coefficients. Second, they ignore valuable information on higher-order correlation (other than the second order correlation) among various dimensions of data.

In this study, we propose a probabilistic factorization model for analyzing polyadic data in a coherent way. Our main contributions are summarized as follows.

- We propose a probabilistic factor model that simultaneously captures factors in all the dimensions of polyadic data. In addition, the joint distribution among all the factors is also captured. The proposed polyadic factorization model is an extension of the aspect model proposed by Hofmann et al [7] from two dimension to multiple-dimensions. We present an EM algorithm for computing the parameters of the proposed model.

- We show that when the KL-divergence is used, the proposed probabilistic polyadic factor model is closely related to the non-negative version of the Tucker model [1] in tensor factorization (NTF) and the given EM algorithm is related to the iterative update algorithms in NTF.

- We present two extensions to our basic framework. First, we show how to incorporate Dirichlet prior to the probabilistic model for the purpose of smoothing. Second, from the NTF interpretation of the factor model, we study the case when the Frobenius norm, instead of the KL-divergence, is used to measure the loss.

- We develop an efficient implementation of the algorithm that takes advantage of the sparseness of data. We prove that the time complexity of our implementation is linear (per iteration) in the number of data records in a dataset.

There are many potential applications to the factor model we proposed, such as extracting and monitoring coherent groups of people and topics in social networks, ranking and recommending important people and documents, as well as summarizing and visualizing data. In this paper, we focus on applying the proposed factor model to an important information retrieval task: personalized recommendation. We first show that our probabilistic factorization model offers a natural way for making personalized recommendations. Then, we use extensive experimental studies on a social bookmarking dataset (Delicious) and a paper citation dataset (CiteSeer) to demonstrate the effectiveness of the proposed model.

The rest of the paper is organized as the following. In Section 2, we describe our factor models in detail. In Section 3, we present two extensions to our basic framework. In Section 4, we discuss some practical issues on efficient computation of the model parameters. In Section 5, we show how the proposed model can be applied to the application of personalized recommendation. In Section 6, we survey related work. We show experimental results in Section 7 and finally give conclusions in Section 8.

2. POLYADIC FACTORIZATION MODEL

In this section, the main theoretical framework of our factor model is described. First, the factor model based on a probabilistic polyadic factorization is presented. Then an interpretation using a non-negative tensor factorization is given. Finally, the connection between the two is proved. For ease of discussion we show the cases of 3-dimensional data in the rest of the paper, although our factor model can be defined on multiple-dimensional data with arbitrary number of dimensions.

2.1 A probabilistic polyadic factor model

We describe the probabilistic framework of our factor model by using the CiteSeer dataset as a motivational example. In the CiteSeer data, each data record is an author-keyword-reference triple. Assume that the dataset contains I distinct authors, J distinct keywords, and K distinct references. Furthermore, assume that there are L latent factors (groups) of authors where each author belongs to these L groups with different probabilities. Note that these L latent factors are not explicit, i.e., they are hidden and need to be learned from training data. A random variable C' is used to represent the prior distribution for the L latent factors of authors. Similarly, it is assumed that there are M latent factors of keywords and N latent factors of references, and random variables C'' and C''' are used to represent the prior distributions of these two sets of latent factors. It is worth noting that the cardinalities of C', C'', and C''' do not have to be identical. In the following discussion, i, j, and k are used to represent the indices for author, keyword, and reference, respectively; l, m and n are used to represent the indices for the factors of author, keyword, and reference, respectively.

With these latent factors defined, the following generative model is used to model how each data record is generated. To generate a data record (i.e., an author-keyword-reference triple $\langle i, j, k\rangle$), a factor c'_l for author, a factor c''_m for keyword, and a factor c'''_n for reference are selected with probability $P(c'_l, c''_m, c'''_n)$. Then depending on the selected factors, an author i is picked with probability $P(i|c'_l)$, a keyword j with probability $P(j|c''_m)$, and a reference k with probability $P(k|c'''_n)$. The assumption of this generative model is that the selections of author, keyword and reference are independent with each other, given the latent factors c'_l, c''_m, c'''_n have been chosen. In the following discussion, to avoid notational clutter, we write $P(c'_l, c''_m, c'''_n)$ as $P(c_{lmn})$, $P(i|c'_l)$ as $P(i|l)$, $P(j|c''_m)$ as $P(j|m)$, and $P(k|c'''_n)$ as $P(k|n)$, respectively.

Furthermore, it is assumed that the data records are generated following an identical independent distribution. By using a_{ijk} to denoted the number of occurrences of the triple $\langle i, j, k\rangle$ in the data, the likelihood of observing all the data records is

$$\prod_{i,j,k}\left[\sum_{l,m,n} P(c_{lmn})P(i|l)P(j|m)P(k|n)\right]^{a_{ijk}}.$$

Then to complete the generative model, the parameters Θ are to be learned in order to maximize the above data likelihood, or equivalently, the logarithm of the data likelihood

$$\arg\max_{\Theta}\sum_{i,j,k} a_{ijk}\log\left[\sum_{l,m,n} P(c_{lmn})P(i|l)P(j|m)P(k|n)\right] \quad (1)$$

where $\Theta = \{P(c_{lmn}), P(i|l), P(j|m), P(k|n)\}\ \forall l, m, n, i, j, k$.

It can be shown that the following EM algorithm can be used to compute the MLE parameters Θ in our generative model. The detailed proof is skipped due to the space limit.

E-Step:

$$\hat{p}_{lmn|ijk} = \frac{P(c_{lmn})P(i|l)P(j|m)P(k|n)}{\sum_{l',m',n'} P(c_{l'm'n'})P(i|l')P(j|m')P(k|n')}, \quad (2)$$

M-Step:

$$P(c_{lmn}) = \sum_{i,j,k} a_{ijk}\hat{p}_{lmn|ijk}, \quad (3)$$

$$P(i|l) = \sum_{j,k,m,n} a_{ijk}\hat{p}_{lmn|ijk} \Big/ \sum_{j,k,l,m,n} a_{ijk}\hat{p}_{lmn|ijk}, \quad (4)$$

$$P(j|m) = \sum_{i,k,l,n} a_{ijk}\hat{p}_{lmn|ijk} \Big/ \sum_{i,k,l,m,n} a_{ijk}\hat{p}_{lmn|ijk}, \quad (5)$$

$$P(k|n) = \sum_{i,j,l,m} a_{ijk}\hat{p}_{lmn|ijk} \Big/ \sum_{i,j,l,m,n} a_{ijk}\hat{p}_{lmn|ijk}. \quad (6)$$

This probabilistic polyadic factor model is related to the aspect model proposed by Hofmann et al [7]. However, it is different from the aspect model in two ways. First, the aspect model is defined on dyadic data (matrices) while our factor model can handle polyadic data with arbitrary number of dimensions. Second, in the aspect model, there is a one-one mapping between the factors in the row space and the factors in the column space while in our model, such a one-one mapping is not enforced. Instead, in our factor model a joint probability, $P(c_{lmn})$, is used to capture the correlation among factors in different dimensions. As a result, our factor model is more flexible and it allows many-to-many mapping between factors in different dimensions.

2.2 A model based on non-negative tensor factorization (NTF)

Now we connect the probabilistic polyadic factor model in the previous section with a non-negative version of the Tucker tensor factorization. For this purpose, we re-interpret the the generative model in a different way. Assume that each author i is encoded with an L-dimensional vector $\vec{x}_i \in \mathcal{R}_+^L$, where $\mathcal{R}_+$ represents non-negative real numbers. Similarly, each keyword j is encoded by $\vec{y}_j \in \mathcal{R}_+^M$ and each reference k by $\vec{z}_k \in \mathcal{R}_+^N$. In addition, a_{ijk} is approximated by a function $f(\vec{x}_i \otimes \vec{y}_j \otimes \vec{z}_k)$ where f belongs to a certain family of functions $\mathcal{F}$ and $\otimes$ denotes the Kronecker product. The objective is to find the best encoding, in terms of $\vec{x}_i$, $\vec{y}_j$, $\vec{z}_k$, and f, that minimizes the following loss:

$$loss = \sum_{i,j,k} dist(a_{ijk}, f(\vec{x}_i \otimes \vec{y}_j \otimes \vec{z}_k)). \quad (7)$$

Now two questions arise: which family of functions $\mathcal{F}$ to choose and what distance function to use. For the first question, we simply choose the family of linear functions, i.e., $f_{\vec{c}}(\vec{x}_i \otimes \vec{y}_j \otimes \vec{z}_k) = \langle \vec{c}, \vec{x}_i \otimes \vec{y}_j \otimes \vec{z}_k \rangle$, where $\langle \cdot, \cdot \rangle$ represents the vector inner product. For the second question, we choose the KL-divergence between a_{ijk} and $f(\vec{x}_i \otimes \vec{y}_j \otimes \vec{z}_k)$, where the KL-divergence is defined by

$$KL(\vec{p}||\vec{q}) \equiv \sum_i p_i \log \frac{p_i}{q_i} - \sum_i p_i + \sum_i q_i. \quad (8)$$

Then, we can rewrite the objective function (7) in terms of tensor factorization as follows. Data is first put into a *data tensor* $\mathcal{A} \in \mathcal{R}_+^{I \times J \times K}$ where $(\mathcal{A})_{ijk} = a_{ijk}$. Next, the vectors $\vec{x}_i$'s are concatenated to a matrix $X \in \mathcal{R}_+^{I \times L}$ where the i-th row of X is the transpose of $\vec{x}_i$. Similarly $\vec{y}_j$'s and $\vec{z}_k$'s are concatenated into matrices $Y \in \mathcal{R}_+^{J \times M}$ and $Z \in \mathcal{R}_+^{K \times N}$. Furthermore, $\vec{c}$ is *folded* into a tensor $\mathcal{C} \in \mathcal{R}_+^{L \times M \times N}$ with $(\mathcal{C})_{lmn} = \vec{c}_{(l-1)MN+(m-1)N+n}$, and we call $\mathcal{C}$ the *core tensor*. After the concatenation and the folding, we have $f_{\vec{c}}(\vec{x}_i \otimes \vec{y}_j \otimes \vec{z}_k) = (\mathcal{C} \times_1 X \times_2 Y \times_3 Z)_{ijk}$ where $\times_1$, $\times_2$, and $\times_3$ are the mode-1, mode-2, and mode-3 multiplications of the tensor $\mathcal{C}$ by the matrices X, Y, and Z, respectively. (Please refer [1] for details about the terminologies.) Then the problem of best encoding becomes the problem of finding the best approximation, in terms of X, Y, Z, and $\mathcal{C}$, that minimizes the following loss

$$loss_{KL} = KL(\mathcal{A}||\mathcal{C} \times_1 X \times_2 Y \times_3 Z). \quad (9)$$

However, there is a tricky issue: because $\mathcal{C} \times_1 X \times_2 Y \times_3 Z = (\mathcal{C} \times_1 Q_X \times_2 Q_Y \times_3 Q_Z) \times_1 (XQ_X^T) \times_2 (YQ_Y^T) \times_3 (ZQ_Z^T)$ for arbitrary orthogonal matrices Q_X, Q_Y, and Q_Z, as a result the solution is not uniquely defined. To make the problem well posed, we add the additional constraint that the columns of X, Y, and Z must all sum to ones.

Equation (9) turns out to be a non-negative version of the Tucker tensor decomposition [14]. By directly extending the algorithm given by Lee and Seung [9], which was used for solving the non-negative matrix factorization (NMF) problem, we have the following iterative update algorithm.

THEOREM 1. *For a given $\mathcal{A} \in \mathcal{R}_+^{I \times J \times K}$, we first define another tensor $\mathcal{B}$ as*

$$\mathcal{B}_{ijk} = \mathcal{A}_{ijk} / (\mathcal{C} \times_1 X \times_2 Y \times_3 Z)_{ijk}, \quad (10)$$

Then the following update rules are guaranteed to converge to an optimal solutions to the objective function defined in Eq. (9).

$$\mathcal{C}_{lmn} \leftarrow \mathcal{C}_{lmn} (\mathcal{B} \times_1 X^T \times_2 Y^T \times_3 Z^T)_{lmn}, \quad (11)$$

$$X_{il} \leftarrow X_{il} \quad \mathcal{B}_{(1)}(Y \otimes Z)\mathcal{C}_{(1)}^T{}_{il}, \quad (12)$$

$$Y_{jm} \leftarrow Y_{jm} \quad \mathcal{B}_{(2)}(Z \otimes X)\mathcal{C}_{(2)}^T{}_{jm}, \quad (13)$$

$$Z_{kn} \leftarrow Z_{kn} \quad \mathcal{B}_{(3)}(X \otimes Y)\mathcal{C}_{(3)}^T{}_{kn}, \quad (14)$$

and normalize s.t. columns of X, Y, Z sum to ones.

We provide the proof for the correctness of this algorithm in the Appendix.

2.3 The equivalence between the two models

It turns out that the model we defined using the NTF framework is *equivalent* to the probabilistic polyadic factor model and therefore, it is just a different interpretation of the probabilistic factor model. We prove this claim in the following theorem.

THEOREM 2. *The problem of minimizing the loss in Equation (9) is equivalent to the MLE problem described by Equation (1).*

PROOF. We first define $\mathcal{D} = \mathcal{C} \times_1 X \times_2 Y \times_3 Z$ and denote $(\mathcal{D})_{ijk}$ by d_{ijk}. Then we can expand the loss function (9) as

$$\begin{aligned} loss_{KL} &= KL(\mathcal{A}||\mathcal{C} \times_1 X \times_2 Y \times_3 Z) \\ &= \sum_{i,j,k} a_{ijk} \log a_{ijk} - a_{ijk} \log d_{ijk} - \sum_{i,j,k} a_{ijk} + \sum_{i,j,k} d_{ijk}. \end{aligned}$$

Several observations can be obtained from the above equation. First, a necessary condition for a solution $\mathcal{D}$ to minimize $loss_{KL}$ is that $\sum_{i,j,k} a_{ijk} = \sum_{i,j,k} d_{ijk}$. This also implies that a necessary condition for an optimal solution is that $\sum_{i,j,k} a_{ijk} = \sum_{l,m,n} (\mathcal{C})_{lmn}$, because of the constraints on X, Y, and Z.

A second observation is $\mathcal{A}$ can be scaled so that $\sum_{i,j,k} a_{ijk} = 1$ without changing the optimization problem. Similarly, because Equation (1) can be multiplied by a constant without affecting the MLE problem, without loss of generality, we can assume that $\sum_{i,j,k} a_{ijk} = 1$. As a result, minimizing $loss_{KL}$ is equivalent to maximizing $\sum_{i,j,k} a_{ijk} \log d_{ijk}$.

By comparing the term $\sum_{i,j,k} a_{ijk} \log d_{ijk}$ with the target function Equation (1) for the MLE problem, and by setting $X_{il} = P(i|l), Y_{jm} = P(j|m), Z_{kn} = P(k|n), (\mathcal{C})_{lmn} = P(c_{lmn})$, we prove the theorem. □

As can be expected, there is a close connection between the EM algorithm in Equations (2)–(6) and the NTF factorization algorithm in Equations (10)–(14). With some simple derivations we can show that the two algorithms share almost the identical form. However, there is a subtle difference between the two. In the EM algorithm, all parameters are updated *together* in the M-step. In comparison, in the NTF factorization algorithm, X, Y, Z, and $\mathcal{C}$ must be updated *alternatively*, which makes it a block descent algorithm.

3. EXTENSIONS TO THE BASIC FRAMEWORK

Having both the probabilistic MLE and NTF interpretations of the factor model is beneficial, because we can further extend the proposed model based on two different perspectives. In this section we present two such kind of extensions to the basic framework.

3.1 Adding Dirichlet prior

One extension is to consider the maximum a posteriori (MAP) estimation for the basic probabilistic model. Here, we consider the Dirichlet distribution as the prior distribution for the parameters.

First, we consider the prior for X. Since $X_{il} = P(i|l)$, where $P(*|l)$ is a multinomial distribution, the prior of each column of X could be a Dirichlet distribution with hyper-parameter α_X[1]. The logarithm of the prior probability is

$$\ln P(X) = \alpha_X \sum_{il} \ln X_{il} + c_X,$$

where c_X is a value irrelevant to X.

Similarly, we assume the priors for Y, Z and $\mathcal{C}$ are all Dirichlet distributions with hyper-parameters α_Y, α_Z and $\alpha_\mathcal{C}$ respectively. The logarithm of the prior probabilities are

$$\begin{aligned}\ln P(Y) =& \alpha_Y \sum_{jm} \ln Y_{jm} + c_Y,\\ \ln P(Z) =& \alpha_Z \sum_{kn} \ln Z_{kn} + c_Z,\\ \ln P(\mathcal{C}) =& \alpha_C \sum_{lmn} \ln \mathcal{C}_{lmn} + c_\mathcal{C}.\end{aligned}$$

The logarithm loss of the MAP estimation is that of MLE plus the logarithm of prior probabilities. Therefore,

$$loss_{MAP} = loss_{KL} - \ln P(X) - \ln P(Y) - \ln P(Z) - \ln P(\mathcal{C}). \tag{15}$$

With some computations, we can derive the following Corollary (for convenience, we chose an expression in NTF).

[1]Dirichlet distribution is a conjugate prior for multinomial distribution.

COROLLARY 1. *The following update rules are guaranteed to converge to an optimal solutions to the objective function defined in Equation* (15).

$$\begin{aligned}\mathcal{C}_{lmn} &\leftarrow \mathcal{C}_{lmn}(\mathcal{B} \times_1 X^T \times_2 Y^T \times_3 Z^T)_{lmn} + \alpha_C,\\ X_{il} &\leftarrow X_{il} \quad \mathcal{B}_{(1)}(Y \otimes Z)\mathcal{C}^T_{(1)\ il} + \alpha_X,\\ Y_{jm} &\leftarrow Y_{jm} \quad \mathcal{B}_{(2)}(Z \otimes X)\mathcal{C}^T_{(2)\ jm} + \alpha_Y,\\ Z_{kn} &\leftarrow Z_{kn} \quad \mathcal{B}_{(3)}(X \otimes Y)\mathcal{C}^T_{(3)\ kn} + \alpha_Z.\end{aligned}$$

and normalize s.t. columns of X, Y, Z sum to ones.

This Dirichlet-prior-based MAP framework is useful in that it provides a way for us to add smoothing to the observed data when the data are sparse. We will demonstrate this in the experimental studies.

3.2 Loss in the Frobenius norm

From the NTF interpretation of the factor model, we can choose any matrix (tensor) norm instead of the KL-divergence to measure the loss. For example, if the Frobenius norm is used, the loss becomes

$$loss_F = \|\mathcal{A} - \mathcal{C} \times_1 X \times_2 Y \times_3 Z\|_F^2 \tag{16}$$

where the Frobenius norm for a tensor $\mathcal{A}$ is defined by $\|\mathcal{A}\|_F^2 = \langle \mathcal{A}, \mathcal{A} \rangle = \sum_{i,j,k} a_{ijk}^2$. For this loss function, we can show that the following algorithm can be used to find an optimal solution.

THEOREM 3. *For a given $\mathcal{A} \in \mathcal{R}_+^{I \times J \times K}$, the following update rules are guaranteed to converge to an optimal solutions to the objective function defined in Equation* (16).

$$\begin{aligned}\mathcal{C}_{lmn} &\leftarrow \mathcal{C}_{lmn} \frac{\mathcal{A} \times_1 X^T \times_2 Y^T \times_3 Z^T{}_{lmn}}{(\mathcal{C} \times_1 (X^T X) \times_2 (Y^T Y) \times_3 (Z^T Z))_{lmn}},\\ X_{il} &\leftarrow X_{il} \frac{\mathcal{A}_{(1)}(Y \otimes Z)\mathcal{C}^T_{(1)\ il}}{X\mathcal{C}_{(1)}((Y^T Y) \otimes (Z^T Z))\mathcal{C}^T_{(1)\ il}},\\ Y_{jm} &\leftarrow Y_{jm} \frac{\mathcal{A}_{(2)}(Z \otimes X)\mathcal{C}^T_{(2)\ jm}}{Y\mathcal{C}_{(2)}((Z^T Z) \otimes (X^T X))\mathcal{C}^T_{(2)\ jm}},\\ Z_{kn} &\leftarrow Z_{kn} \frac{\mathcal{A}_{(3)}(X \otimes Y)\mathcal{C}^T_{(3)\ kn}}{Z\mathcal{C}_{(3)}((X^T X) \otimes (Y^T Y))\mathcal{C}^T_{(3)\ kn}},\end{aligned}$$

and normalize s.t. columns of X, Y, Z sum to ones.

This algorithm is a direct extension of an algorithm by Lee and Seung [9] for NMF and we provide the proof for the correctness of this algorithm in the Appendix.

4. IMPLEMENTATION DETAILS AND TIME COMPLEXITY

In this section, we describe some practical details in the implementation of our factor models and provide time complexity analysis.

4.1 Implementation details

In the following discussion, without loss of generality, it is assumed that $L \geq M \geq N$ and $I \geq J \geq K$. We assume that there are nz data records in the dataset, which implies that

there are nz non-zero entries in the data tensor $\mathcal{A}$. In addition, we assume that there are np distinct (i, j) pairs in the dataset. For ease of discussion, we use the NTF framework in the following discussion, although the same ideas apply to the EM algorithm.

As a matter of fact, most components in Equations (10)–(14), especially those time-consuming parts, can be computed in the same computational framework. We describe this computation framework by showing how to compute $\mathcal{B}$ in Equation (10), while other computations can be conducted in a similar fashion. A naive implementation for computing $\mathcal{B}$ will expand $\mathcal{C} \times_1 X \times_2 Y \times_3 Z$, which turns out to be a dense tensor in $\mathcal{R}_+^{I \times J \times K}$. However, because such a dense tensor contains IJK entries, it is impractical to make such an explicit expansion (e.g., consider a dataset with 10K authors, 10K keywords, and 10K references, where $IJK = 1$ trillion). We design an implementation that takes advantage of the sparseness of the dataset and has a time complexity that is linear (per iteration) in nz, the number of non-zero entries in the raw data tensor $\mathcal{A}$.

Assuming each data record of the data is stored in the format of $\langle key1, key2, key3, v\rangle$, where $key1, key2$ and $key3$ are the indices of the data record in the first, second, and third dimensions, respectively; v is the value of the data record, which can be, for example, the number of times that an author cites a reference on a keyword. In the first step of the algorithm, the data records are sorted according to $key1$ as the major key and then $key2$ and then $key3$ as the minor keys. This sorting takes time linear in nz because the keys are integers with known upper-bounds and therefore a linear sorting algorithm, such as bucket sort, can be applied. In addition, to simplify the discussion, it is assumed that $\langle key1, key2, key3\rangle$ is a primary (unique) key for the data records.

There are two key ideas in our implementation. First, our implementation is a *lazy* one where only the absolute necessary computations are performed. In such a lazy fashion, the implementation takes advantage of the sparseness of the data and only performs computations on entries corresponding to non-zero values in the dataset. The second key idea of our implementation is that the computations are carefully ordered in a way such that repeated computations are minimized. Figure 1 illustrates these two key ideas using pseudo-codes. In the algorithm, D is an $M \times N$ matrix and $\vec{d}$ is a vector of length N. Note that $fold(\cdot)$ is a function for folding a vector into a matrix. This function does not have to be materialized in the real implementation and is shown here only to make it easier to understand. The idea of lazy computation is reflected by the fact that the *for* loop enumerates only non-zero entries in the dataset (or equivalently, in $\mathcal{A}$). To see the idea of ordered computation, we note that the D matrix computed at line 5 for a given i is usually re-used several times at line 8 for different j's, and the $\vec{d}$ computed at line 8 is usually re-used several times at line 10 for different k's.

4.2 Time complexity

It can be easily seen that the algorithm in Figure 1 has a time complexity of $O(nz \cdot L + np \cdot L^2 + I \cdot L^3)$, where nz is the number of data records and np is the number of distinct (i, j) pairs in the dataset. Here are some observations on the time complexity:

- If we consider L, the number of factors, as a constant, then the time is linear in nz. This characteristic is good because datasets in real applications often have high cardinality in each dimension but are sparse overall and therefore, an algorithm linear in nz is practically appealing.
- If we do not consider L as a constant, then some terms in the time complexity are proportional to L^3, L^2, and L, respectively. However, we have $nz > np > I$. When the difference between nz, np, and I is large, which is very common in real applications where data distributions are often skewed, the $np \cdot L^2$ term or even the $nz \cdot L$ term dominates the running time instead of the $I \cdot L^3$ term.
- An additional good property of our algorithm is that because the data records are accessed in a sequential way and because in each iteration, the computation on one data record does not depend on the computation on other data records, our algorithm can be easily parallelized.

Algorithm for computing $\mathcal{B}$

```
    input: data records ⟨key1, key2, key3, v⟩
    output: B
1:  i ← −1, j ← −1;
2:  for each data record ⟨key1, key2, key3, v⟩ do
3:      if i ≠ key1
4:          i ← key1, j ← −1;
5:          D ← fold(X_{row_i} · C_(1));
6:      if j ≠ key2
7:          j ← key2;
8:          d⃗ ← Y_{row_j} · D;
9:      k ← key3;
10:     b_ijk ← v/(Z_{row_k} · d⃗);
11: return B;
```

Figure 1: Computing Tensor $\mathcal{B}$

4.3 Computing the loss

Our algorithms are iterative updating ones and therefore the losses in Equations (9) and (16) are needed in each iteration to check convergence. Again, to explicitly expanding $\mathcal{C} \times_1 X \times_2 Y \times_3 Z$ is impractical. Here we give details on how we efficiently compute the two losses.

The loss in the KL-divergence can be easily computed while computing

$$\mathcal{B}_{ijk} = \mathcal{A}_{ijk} / \left(\mathcal{C} \times_1 X \times_2 Y \times_3 Z\right)_{ijk}$$

at no extra cost. The loss in the Frobenius norm is more tricky. We first rewrite the loss in the Frobenius norm as the following

$$\begin{aligned}
&\|\mathcal{A} - \mathcal{C} \times_1 X \times_2 Y \times_3 Z\|_F^2 \\
&= \langle \mathcal{A} - \mathcal{C} \times_1 X \times_2 Y \times_3 Z, \mathcal{A} - \mathcal{C} \times_1 X \times_2 Y \times_3 Z\rangle \\
&= \langle \mathcal{A}, \mathcal{A}\rangle + \langle \mathcal{C}, \mathcal{C} \times_1 (X^T X) \times_2 (Y^T Y) \times_3 (Z^T Z)\rangle \\
&- 2\langle \mathcal{A}, \mathcal{C} \times_1 X \times_2 Y \times_3 Z\rangle
\end{aligned}$$

In the last equation, the first term is only related to the non-zero entries in $\mathcal{A}$ and so can be computed with time linear in nz; the second term involves an inner product between two tensors with small sizes in all the dimensions and so it

can be handled quickly; the third term has a form similar to the computation in the update algorithm that has been shown in Figure 1, and therefore the time complexity for computing this term is again $O(nz \cdot L + np \cdot L^2 + I \cdot L^3)$. In summary, computing the losses does not change the total time complexity of the algorithms.

5. APPLICATION TO PERSONALIZED RECOMMENDATION

There can be many applications using our factor models, such as discovering coherent groups in social networks, summarizing multiple-dimensional data, ranking and recommending important people and documents, etc. In this paper, we focus on an important information retrieval task: personalized recommendation.

We again use the CiteSeer dataset as a motivation example where the task is to recommend the most relevant references to a given author for a given keyword. A straightforward solution to this problem is to simply recommend the most popular references associated with the given keyword. Such an approach, however, makes global recommendations to all the authors and therefore ignores any personal information about the author to whom the recommendation is made. Personal information can be very useful for recommendations because different authors have different areas of expertise and focuses of research. So for a given keyword, the most globally popular references on the keyword may not be the most relevant ones for a particular author.

Our polyadic factor model provides a natural way to incorporate an author's background in order to make personalized recommendation. Given a probabilistic model, for a given author i and a given keyword j, a reasonable solution is to recommend the references k's that have the highest conditional probabilities predicted by the model. In the naive method based on the global popularity, the conditional probability is only conditioned on the keyword j, i.e., the naive method uses $P(k|j)$ for recommendation. In comparison, in our probabilistic polyadic factor model, the conditional probability $P(k|i,j)$ that is conditioned on both the author i and the keyword j is used for *personalized* recommendation, where

$$P(k|ij) = \frac{P(ijk)}{P(ij)} \propto P(ijk) = \mathcal{C} \times_1 X_{row_i} \times_2 Y_{row_j} \times_3 Z_{row_k}$$
$$= Z_{row_k} \mathcal{C}_{(3)} (X^T_{row_i} \otimes Y^T_{row_j}).$$

6. RELATED WORK

One research area that is closely related to our work is general tensor factorizations where the non-negative constraint is removed. Two examples of such tensor decompositions are the Tucker model [14], which has recently been generalized to the Higher Order Singular Value Decomposition [1]) and the PARAFAC model [5]. These tensor factorization techniques show some good properties in terms of minimizing certain losses in the Frobenius norm. However, the factors and core tensors obtained by these general factorizations have negative values, which make them difficult to interpret. In addition, factors obtained from these tensor factorizations usually have some orthogonality constraints. All of these restrict the applications of these general tensor factorizations in the IR field.

As we have mentioned before, our factor model for polyadic data is closely related to the aspect model (PLSA) proposed by Hofmann et al [7], which can be solved by the non-negative matrix factorization (NMF) algorithms proposed by Lee and Seung [9]. The MLE interpretation of our factor model is an extension to PLSA and the NTF interpretation is an extension to NMF. Recently, Gaussier and Goutte [4] and Ding et.al. [3] have shown the equivalence between the PLSA and the NMF algorithms. Therefore it comes at no surprise that the MLE and the NTF interpretations of our factor model are equivalent. Also, we are not the first ones to notice that the NMF algorithms can be directly extended to solving NTF problems. For example, Mørup et al [11] have explored this connection and developed similar algorithms. In addition, there also exist non-negative versions of the PARAFAC model [6, 12]. However, a weak point of such algorithms is that they require the number of factors in each dimension to be identical and the core tensor to be diagonal. This limits the usage of the model in many applications.

Personalized recommender systems has been studied for many years in three sub areas in IR: relevance feedback, adaptive filtering and collaborative filtering. **Adaptive filtering** and **relevance feedback** focus on identifying topically relevant documents based on the *content* of the documents and update the user profile using content based retrieval models (e.g. Boolean models, vector space models, traditional probabilistic models, inference networks and language models) or machine learning algorithms (e.g. Support Vector Machine (SVM), K nearest neighbors (K-NN), neural network, logistic regression and Winnow). **Collaborative filtering** recommends items to a user using information from other users with similar tastes and preferences in the past. Memory based heuristics and model based approaches have been used in collaborative filtering tasks [2, 8, 13]. However, collaborative filtering algorithms usually only use two dimensional data represented as user-item matrix. The model proposed in this paper enables us to go beyond the user-item matrix and use more information, such as social networks, for better personalized recommendation.

7. EXPERIMENTAL STUDIES

7.1 Datasets and tasks

In this section, we apply our factor models to the problem of personalized recommendation and study the performance by using two datasets. The first dataset is obtained from the social bookmarking website Del.icio.us[2]. At Del.icio.us, registered *users* assign *tags* to *urls*. Therefore, data records in this dataset, which we call *Delicious*, consist of *user-tag-url* triples. In our study, we first preprocess the data by removing users, tags, and urls that have occurrences less than a threshold of 20. This results a Delicious dataset with 3,908 users, 867 tags, 3,570 urls, and 180,246 data records. Figure 2 shows the histograms of the data. As can be seen, users, tags, and urls all clearly follow power-law distributions. On the Delicious dataset we define two tasks of personalized recommendation. In the first task, for a given url, relevant tags are recommended to a particular user. This task is directly applicable to personalized tag recommendation in social bookmarking systems such as Del.icio.us. The

[2]http://del.icio.us/

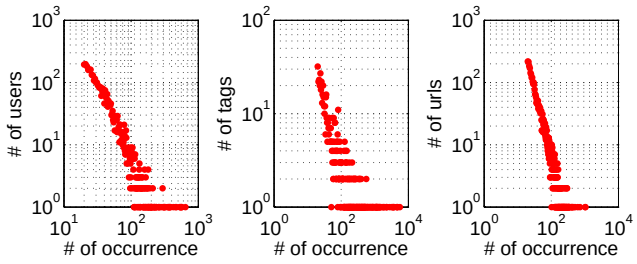

Figure 2: The histograms for users, tags, and urls in the Delicious dataset.

second task is in a dual form of the first one: for a given tag, relevant urls are recommended to a particular user. This task can be of importance in a personalized search engine. That is, if a tag is considered as a query, then the collaborative tagging reflects people's tendency to associate different urls to the query and the history of a particular user reflects that user's preference among the urls for the given query.

The second dataset is obtained from the CiteSeer website[3]. A set of articles from various areas are selected first. The author(s), abstract, and references of each article are extracted and the abstracts are split into keywords. Finally the *author-keyword-reference* triples are extracted. Similar preprocessing process is applied to the CiteSeer data, which results in a dataset, which we call *CiteSeer*, containing 16,466 authors, 920 keywords, 29,309 references, and 3,636,020 data records. On the CiteSeer dataset, we define the task as recommending relevant references for a given author on a given keyword. This task is useful for an author to collect important references when writing an article on a particular topic.

7.2 Performance measure and baselines

For the performance studies, data are randomly split into 80% training data and 20% testing data. However, a subtle issue here is that the split should not be totally random. To illustrate this issue, we take the task of tag recommendation as an example. For this task, for fair comparison, the training and testing data should be split in such a way that a user-url pair either shows up in the training data or in the testing data, but not in both. In other words, if we measure the performance by how good the recommended tags are on a given url for a particular user, then the user should not have seen the url in the training data. As a result, for different tasks, the data are split in slightly different ways.

Factorizations are applied to the training data to get the parameters for our factor models, i.e., the factors in each dimensions and their correlation. Then the learned factor models are applied to the testing data for top-K personalized recommendation with various K's. The recommendation performance is measured by the average Normalized Discounted Cumulative Gain (NDCG) where NDCG is defined as

$$NDCG = \sum_j \frac{2^{rel(j)} - 1}{\log(1+j)}.$$

NDCG is a relatively new measure used extensively in Web searches. It allows different levels of relevance and weighs the recommendations according to their ranks in the ranked list. In our study, we use a binary relevance score for $rel(j)$ where $rel(j) = 1$ if item j occurs in the testing data and 0 otherwise. More specifically, we use top-3 tag recommendation as an example to illustrate how the average NDCG is computed. Assume that for a user i on a url k, our model recommends an ordered list of three tags $\{j_1, j_2, j_3\}$. Furthermore, assume that $\langle i, j_1, k\rangle$ and $\langle i, j_3, k\rangle$ show up in the testing data but $\langle i, j_2, k\rangle$ does not. Then the NDCG for the (i, k) pair is $1/\log(1+1) + 1/\log(3+1) = 1.5$. The average NDCG is the NDCG scores averaged over all distinct (i, k) pairs in the testing data.

We compare the performance of our factor models with that of two baseline algorithms. These baseline algorithms offer *global* rather than *personalized* recommendations. We explain these baseline algorithms using the task of tag recommendation. First, the user-tag-url triples are aggregated into tag-url pairs together with the count of each pair. The aggregated results are put into a matrix M where the rows and columns of M correspond to tags and urls, respectively, and the jk-th entry in M represents the count of occurrences of tag j associated with url k. For the first baseline algorithm, the recommendation is only based on the count of occurrences. That is, for url k, this baseline recommends the tags with highest magnitudes in the k-th column of M. This baseline algorithm corresponds to recommending by popularity and such a straightforward method usually shows very good performance when enough observations are sampled from data. In the second baseline algorithm, a non-negative matrix factorization (NMF) is applied to M to compute an approximation $\tilde{M}$ to M. After $\tilde{M}$ is obtained, the recommendations are made in a similar way to the first baseline, except that $\tilde{M}$ is used instead of M. The intuition behind this baseline is similar to the intuition behind the latent semantic indexing (LSI) — by capturing the major components from the raw data, we may be able to reduce the noise in the raw data and therefore discover the hidden relations.

7.3 Results on the Delicious dataset

In the first task, namely recommending relevant tags to a given user on a given url, our factor models do not perform as well as the baseline algorithms. Table 1 gives the average NDCG scores for different algorithms. For our factor models, 100 factors are used in each of the three dimensions. We believe that the main reason for the poor performance of the factor models is that personalization does not help in this task. That is, different users usually have consensus on the tags for a url. For example, for a homepage of a professor whose research is on IR, most likely a user will assign tags such as "IR" and "people", despite the background of the user. Another observation from the results is that because the data are sparse, adding smoothing (the case reported as

	top-1	top-3	top-5	top-10
baseline1	**0.699**	**1.308**	**1.622**	**1.879**
baseline2	0.636	1.192	1.473	1.747
Factor_KL_100	0.217	0.397	0.481	0.602
Factor_KL_100_S	0.384	0.688	0.833	1.018
Factor_F_100	0.196	0.355	0.445	0.564

Table 1: Performance of different algorithms for the task of recommending tags on Delicious dataset.

[3]http://citeseer.ist.psu.edu/

Factor_KL_100_S in Table 1, with α's set to 0.001) improves the performance over the case where no smoothing is used (the case reported as Factor_KL_100 in Table 1).

On the other hand, for the second task, namely recommending relevant urls to a given user on a given tag, personalization does help. As can be see from Table 2, the factor model using the KL-divergence has better performance than the baseline algorithms. Intuitively, on the same tag, different users may have different ideas on what urls are relevant, depending on the user's interests and background and as a result, personalization offers certain help in this task.

	top-1	top-3	top-5	top-10
baseline1	0.091	0.149	0.175	0.208
baseline2	0.046	0.076	0.090	0.111
Factor_KL_100	**0.102**	**0.171**	**0.204**	**0.251**
Factor_F_100	0.100	0.131	0.142	0.158

Table 2: Performance of different algorithms for the task of recommending urls on Delicious dataset.

Another observation from Tables 1 and 2 is the factor model that uses KL-divergence to measure the loss clearly outperforms the one that uses Frobenius norm to measure the loss. Puzzled by this discrepancy, we investigated the factors obtained by the two models. It turns out, as shown in Figure 3, using Frobenius norm to measure the loss will result in factors that are much more sparser than those obtained by using KL-divergence. This result is because that Frobenius norm focuses heavily on entries with large values in the raw data while KL-divergence focuses on all entries relatively evenly. This difference in the resulting factors suggests that KL-divergence is a more appropriate loss measure when the task is recommending or ranking.

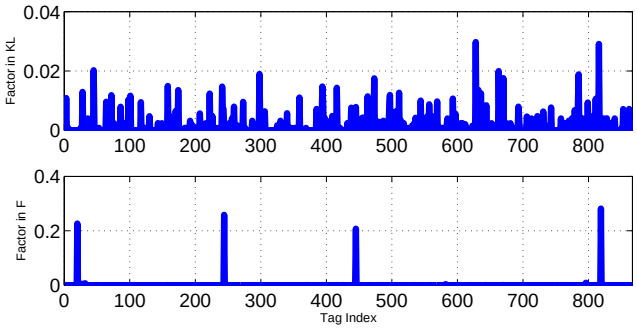

Figure 3: A typical tag factor obtained by using KL divergence (upper) and a typical tag factor obtained by using the Frobenius norm (lower).

7.4 Results on the CiteSeer dataset

For the CiteSeer dataset, the main task is to recommend relevant references to a given author on a given keyword. Table 3 shows the performance in average NDCG scores. For the factor model using KL-divergence, 10 factors are used for each of the three dimensions while for the factor model using Frobenius norm, we report two cases where 10 factors and 30 factors are used respectively. As can be seen from the results, in this task our factor models are able to outperform the baseline algorithms using very small number of factors (10 for the model using KL-divergence and 30 for the model using Frobenius norm).

	top-1	top-3	top-5	top-10
baseline1	0.025	0.045	0.057	0.076
baseline2	0.021	0.037	0.047	0.062
Factor_KL_10	**0.038**	**0.070**	**0.090**	**0.120**
Factor_F_10	0.021	0.040	0.047	0.062
Factor_F_30	0.025	0.048	0.061	0.076

Table 3: Performance of different algorithms for the task of recommending references on CiteSeer data.

This good performance suggests that this dataset, CiteSeer, is "easier" to handle than the pervious Delicious data. It is easy in the sense that personalization tells us a lot about an author. To validate this point, we show in Tables 4, 5, and 6 the top members in some factors obtained by our factor model by using KL-divergence. As can be seen, the factors obviously form clusters of authors focusing on different research areas with different relevant references. In comparison, top members in the factors obtained from the Delicious dataset (not shown due to the space limit) do not show such clear clusters. It is worth mentioning that these factors for the three dimensions are *simultaneously* obtained by our factor models from the polyadic data. That is, our factor models are effective in simultaneously analyzing all dimensions as well as their correlations among polyadic data. In addition, these factors and their correlations can be very useful for other tasks such as clustering and summarization.

DB & DM	Networks	AI & ML
T Eiter	J L Boudec	S Thrun
G Karypis	D Estrin	W Burgard
G D Giacomo	Z Zhang	H Zhang
W Faber	J A Stankovic	J Malik
N Leone	V Firoiu	M J Black
E Keogh	K G Shin	J A Fessler
J Yang	G B Giannakis	M J Mataric
J Han	J Liebeherr	C Sanderson
V J Tsotras	T Abdelzaher	R Deriche
D Calvanese	C Lu	D Fox
D Zhang	D Towsley	S Belongie
W Wang	I Stojmenovic	J Huang
M Hanus	T He	S Z Li
H Garcia-Molina	G Manimaran	I Cohen
S Mehrotra	E W Knightly	G Sapiro
D Gunopulos	M Reisslein	S Soatto
C Zaniolo	J Heidemann	A M Tekalp
K Chakrabarti	C Li	A S Willsky
L Bertossi	K Lam	M Gales
E A Rundensteiner	A Helmy	T Poggio

Table 4: Top members in some author factors

7.5 Results on running time

We designed several experiments to study the performance of our implementation in terms of running time. For the purpose of experimental study, we generate a second Delicious dataset, which we call *Delicious-2*, by using a lower threshold (10) in the pre-processing steps which results in dataset that is 3 times larger (with 482,600 data records) than the previous Delicious dataset (which we refer to as *Delicious-1*).

The factor model based on KL-divergence is applied to the two datasets over a variety of factor numbers ranging from

DB & DM	Networks	AI & ML
database	network	learning
documents	scheduling	planning
information	protocol	evolutionary
document	differentiated	reinforcement
language	mobility	dialogue
semantic	location	boosting
approach	packets	combinatorial
semi-structured	capacity	classification
disjunctive	channels	satisfaction
declarative	communication	optimization
extraction	schedulers	approach
programs	differentiation	negotiation
representation	multiplexing	bayesian
dimensionality	queueing	function
datasets	support	planners

Table 5: Top members in some keyword factors

1.	Fast algorithms for mining association rules
2.	Mining association rules between sets of items in large databases
3.	The anatomy of a large-scale hypertextual Web search engine
4.	Authoritative sources in a hyperlinked environment
5.	The stable model semantics for logic programming
6.	Information retrieval
1.	Dynamic source routing in ad hoc wireless networks
2.	A performance comparison of multi-hop wireless ad hoc network routing protocols
3.	Ad-hoc on-demand distance vector routing
4.	Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers
5.	Next century challenges: scalable coordination in sensor networks
6.	A highly adaptive distributed routing algorithm for mobile wireless networks
1.	Reinforcement learning I: introduction
2.	Bagging predictors
3.	Support-vector networks
4.	Optimization by simulated annealing
5.	A decision-theoretic generalization of on-line learning and an application to boosting
6.	Experiments with a new boosting algorithm

Table 6: Top members in some reference factors

10 to 50. First, the running time is shown in Figure 4(a) in a log-log scale. As can be seen, the running time scales in a polynomial fashion with respect to L, the number of factors in each dimensions. From the slope of the curves in Figure 4(a), it can be shown that the running time is proportional to L^2. Second, it can be seen that the ratio between the running time on the two datasets, which is given in Figure 4(b), remains a constant 3, which is about the ratio between the sizes of the two datasets. All these results validate our theoretical time complexity analysis.

8. CONCLUSION

In this paper, we propose a polyadic factor model to analyze multiple-dimensional data such as networked data in social networks. This probabilistic generative model is related to non-negative tensor factorizations. Based on the probabilistic and NTF interpretation of the model, we propose two extensions to the basic framework. The factor model can be learned efficiently with a time complexity linear to the number of data records. This model has many potential applications. As an example, we apply the polyadic factor

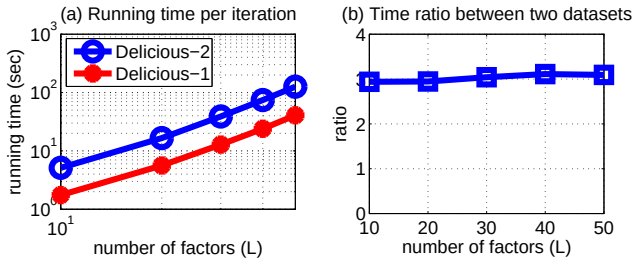

Figure 4: (a) The running time per iteration for the two Delicious datasets, (b) The ratio of running time between the two Delicious datasets.

model to the task of personalized recommendation, and obtain encouraging experimental results on social bookmarking dataset *Del.icio.us* and paper citation dataset *CiteSeer*.

9. ACKNOWLEDGMENTS

We thank Professor C. Lee Giles for providing us the CiteSeer dataset and Haward Jie for sharing the De.li.cio.us data. The last author's research is sponsored by National Science Foundation IIS-0713111. Any opinions, findings, conclusions or recommendations expressed in this paper do not necessarily reflect those of the sponsor.

10. REFERENCES

[1] L. De Lathauwer, B. De Moor, and J. Vandewalle. A multilinear singular value decomposition. *SIAM J. on Matrix Analysis and Applications*, 21(4), 2000.

[2] J. Delgado and N. Ishii. Memory-based weightedmajority prediction for recommender systems. In *ACM SIGIR'99 Workshop on Recommender Systems*, 1999.

[3] C. Ding, X. He, and H. D. Simon. On the equivalence of nonnegative matrix factorization and spectral clustering. In *SIAM SDM*, 2005.

[4] E. Gaussier and C. Goutte. Relation between plsa and nmf and implications. In *SIGIR*, 2005.

[5] R. A. Harshman. Foundations of the parafac procedure: models and conditions for an "explanatory" multi-modal factor analysis. *UCLA working papers in phonetics*, 16, 1970.

[6] T. Hazan, S. Polak, and A. Shashua. Sparse image coding using a 3d non-negative tensor factorization. In *ICCV '05: Proceedings of the Tenth IEEE International Conference on Computer Vision*, 2005.

[7] T. Hofmann, J. Puzicha, and M. I. Jordan. Learning from dyadic data. In *NIPS 11*, 1999.

[8] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl. GroupLens: Applying collaborative filtering to Usenet news. *Communications of the ACM*, 40(3):77–87, 1997.

[9] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *NIPS*, 2000.

[10] Q. Mei, D. Cai, D. Zhang, and C. Zhai. Topic modeling with network regularization. In *Proc. of the 17th WWW Conference*, 2008.

[11] M. Mørup, L. K. Hansen, and S. M. Arnfred. Algorithms for sparse non-negative TUCKER (also

named HONMF). Technical report, Department of Informatics and Mathematical Modeling, Technical University of Denmark, 2007.

[12] 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*, 2005.

[13] L. Si and R. Jin. A flexible mixture model for collaborative filtering. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML 2003)*, 2003.

[14] L. R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31, 1966.

[15] D. Zhou, S. Zhu, K. Yu, X. Song, B. L. Tseng, H. Zha, and C. L. Giles. Learning multiple graphs for document recommendations. In *WWW*, 2008.

[16] S. Zhu, K. Yu, Y. Chi, and Y. Gong. Combining content and link for classification using matrix factorization. In *SIGIR*, 2007.

APPENDIX

A. PROOF FOR THEOREM 1

PROOF. We start with the proof for the update rule for X. In the proof, we use $\tilde{X}$, $\tilde{Y}$, $\tilde{Z}$, $\tilde{\mathcal{B}}$, and $\tilde{\mathcal{C}}$ to represent the *current* values of X, Y, Z, $\mathcal{B}$ and $\mathcal{C}$ before the update. With Y, Z and $\mathcal{C}$ fixed to the values $\tilde{Y}$, $\tilde{Z}$ and $\tilde{\mathcal{C}}$, we consider $KL(\mathcal{A}||\tilde{\mathcal{C}} \times_1 X \times_2 \tilde{Y} \times_3 \tilde{Z})$ as a function of X, i.e., $D_X(X)$. In addition, to avoid notation clutter, we define

$$\theta_{ijklmn} = \tilde{\mathcal{C}}_{lmn}\tilde{X}_{il}\tilde{Y}_{jm}\tilde{Z}_{kn} / \left[\tilde{\mathcal{C}} \times_1 \tilde{X} \times_2 \tilde{Y} \times_3 \tilde{Z}\right]_{ijk}.$$

Then we have

$$\begin{aligned}
&D_X(X)\\
=&\sum_{ijk}\left(\sum_{lmn}\tilde{\mathcal{C}}_{lmn}X_{il}\tilde{Y}_{jm}\tilde{Z}_{kn} - \mathcal{A}_{ijk}\ln(\sum_{lmn}\tilde{\mathcal{C}}_{lmn}X_{il}\tilde{Y}_{jm}\tilde{Z}_{kn})\right) + c_1\\
\leq&\sum_{ijklmn}\tilde{\mathcal{C}}_{lmn}X_{il}\tilde{Y}_{jm}\tilde{Z}_{kn} - \theta_{ijklmn}\mathcal{A}_{ijk}\ln\frac{\tilde{\mathcal{C}}_{lmn}X_{il}\tilde{Y}_{jm}\tilde{Z}_{kn}}{\theta_{ijklmn}} + c_1\\
=&\sum_{ijklmn}\tilde{\mathcal{C}}_{lmn}X_{il}\tilde{Y}_{jm}\tilde{Z}_{kn}\\
&-\tilde{\mathcal{B}}_{ijk}\tilde{\mathcal{C}}_{lmn}\tilde{X}_{il}\tilde{Y}_{jm}\tilde{Z}_{kn}\ln(\tilde{\mathcal{C}}_{lmn}X_{il}\tilde{Y}_{jm}\tilde{Z}_{kn}) + c_2\\
=&\sum_{lmn}\tilde{\mathcal{C}}_{lmn} - \sum_{ijklmn}\tilde{\mathcal{B}}_{ijk}\tilde{\mathcal{C}}_{lmn}\tilde{X}_{il}\tilde{Y}_{jm}\tilde{Z}_{kn}\ln(\tilde{\mathcal{C}}_{lmn}X_{il}\tilde{Y}_{jm}\tilde{Z}_{kn}) + c_2\\
\doteq&Q_X(X;\tilde{X}).
\end{aligned}$$

Therefore, $Q_X(X;\tilde{X})$ is an *auxiliary function* of $D_X(X)$ in the sense that

1. $D_X(X) \leq Q_X(X;\tilde{X})$, and
2. $D_X(X) = Q_X(X;X)$.

So the problem is reduced to minimizing $Q_X(X;\tilde{X})$ with respect to X, under the constraint that all the columns of X sum to ones. We define the Lagrangian

$$L(X,\vec{\alpha}) = Q(X;\tilde{X}) + \vec{\alpha}^T(X^T\vec{1}_I - \vec{1}_L),$$

and by taking its derivative and setting the result to zero, we have

$$\begin{aligned}
\frac{\partial L}{\partial X_{il}} &= \frac{\tilde{X}_{il}}{X_{il}}\sum_{jkmn}\tilde{\mathcal{B}}_{ijk}\tilde{\mathcal{C}}_{lmn}\tilde{Y}_{jm}\tilde{Z}_{kn} + \alpha_l = 0\\
\frac{\partial L}{\partial \alpha_l} &= \sum_i X_{il} - 1 = 0
\end{aligned}$$

Applying similar procedures to Y, Z, and $\mathcal{C}$ we have

$$\begin{aligned}
\frac{\partial L}{\partial Y_{jm}} &= \frac{\tilde{Y}_{jm}}{Y_{jm}}\sum_{ikln}\tilde{\mathcal{B}}_{ijk}\tilde{\mathcal{C}}_{lmn}\tilde{X}_{il}\tilde{Z}_{kn} + \beta_m = 0\\
\frac{\partial L}{\partial \beta_m} &= \sum_j Y_{jm} - 1 = 0\\
\frac{\partial L}{\partial Z_{kn}} &= \frac{\tilde{Z}_{kn}}{Z_{kn}}\sum_{ijlm}\tilde{\mathcal{B}}_{ijk}\tilde{\mathcal{C}}_{lmn}\tilde{X}_{il}\tilde{Y}_{jm} + \gamma_n = 0\\
\frac{\partial L}{\partial \gamma_n} &= \sum_k Z_{kn} - 1 = 0\\
\frac{\partial L}{\partial \mathcal{C}_{lmn}} &= 1 - \frac{\tilde{\mathcal{C}}_{lmn}}{\mathcal{C}_{lmn}}\sum_{ijk}\tilde{\mathcal{B}}_{ijk}\tilde{X}_{il}\tilde{Y}_{jm}\tilde{Z}_{kn} = 0
\end{aligned}$$

which give the update rules in Theorem 1. □

B. PROOF FOR THEOREM 3

PROOF. We proof the update rules for X and $\mathcal{C}$. In addition, in the proof, we use $\tilde{X}$, $\tilde{Y}$, $\tilde{Z}$, and $\tilde{\mathcal{C}}$ to represent the *current* values of X, Y, Z, and $\mathcal{C}$ before the update.

With Y, Z and $\mathcal{C}$ fixed to the values $\tilde{Y}$, $\tilde{Z}$ and $\tilde{\mathcal{C}}$, we consider $\|\mathcal{A} - \tilde{\mathcal{C}} \times_1 X \times_2 \tilde{Y} \times_3 \tilde{Z}\|_F^2$ as a function of X, i.e., $D_X(X)$. In addition, by unfolding the tensor with respect to the first mode, we have

$$D_X(X) = \|\mathcal{A}_{(1)} - X\tilde{\mathcal{C}}_{(1)}(\tilde{Y}^T \otimes \tilde{Z}^T)\|_F^2$$

Then we can apply the results by Lee and Seung [9] to get the update rule

$$\begin{aligned}
X_{il} &\leftarrow X_{il}\frac{\mathcal{A}_{(1)}(Y\otimes Z)\mathcal{C}_{(1)}^T{}_{il}}{X\mathcal{C}_{(1)}\left((Y^T\otimes Z^T)(Y\otimes Z)\right)\mathcal{C}_{(1)}^T{}_{il}}\\
&= X_{il}\frac{\mathcal{A}_{(1)}(Y\otimes Z)\mathcal{C}_{(1)}^T{}_{il}}{X\mathcal{C}_{(1)}\left((Y^TY)\otimes(Z^TZ)\right)\mathcal{C}_{(1)}^T{}_{il}}
\end{aligned}$$

which give the update rule for X in Theorem 3.

For the update rule for $\mathcal{C}$, we rewrite the objective function as

$$D_{\mathcal{C}}(\mathcal{C}) = \|vec(\mathcal{A}) - (\tilde{Z}\otimes\tilde{Y}\otimes\tilde{X})vec(\mathcal{C})\|_F^2$$

Then again we can apply the results by Lee and Seung [9] to get the update rule

$$vec(\mathcal{C})_i \leftarrow vec(\mathcal{C})_i\frac{(Z^T\otimes Y^T\otimes X^T)vec(\mathcal{A})_i}{((Z^T\otimes Y^T\otimes X^T)(Z\otimes Y\otimes X)vec(\mathcal{C}))_i}$$

which give the update rule for $\mathcal{C}$ in Theorem 3. □