FacetCube: A Framework of Incorporating Prior Knowledge into Non-negative Tensor Factorization

Yun Chi
NEC Laboratories America, Inc.
10080 North Wolfe Road, SW3-350
Cupertino, CA 95014, USA
ychi@sv.nec-labs.com

Shenghuo Zhu
NEC Laboratories America, Inc.
10080 North Wolfe Road, SW3-350
Cupertino, CA 95014, USA
zsh@sv.nec-labs.com

ABSTRACT

Non-negative tensor factorization (NTF) is a relatively new technique that has been successfully used to extract significant characteristics from polyadic data, such as data in social networks. Because these polyadic data have multiple dimensions (e.g., the author, content, and timestamp of a blog post), NTF fits in naturally and extracts data characteristics *jointly* from different data dimensions. In the standard NTF, all information comes from the observed data and end users have no control over the outcomes. However, in many applications very often the end users have certain prior knowledge, such as the demographic information about individuals in a social network or a pre-constructed ontology on the contents, and therefore prefer the extracted data characteristics being consistent with such prior knowledge. To allow users' prior knowledge to be naturally incorporated into NTF, in this paper we present a novel framework—FacetCube—that extends the standard non-negative tensor factorization. The new framework allows the end users to control the factorization outputs at three different levels for each of the data dimensions. The proposed framework is intuitively appealing in that it has a close connection to the probabilistic generative models. In addition to introducing the framework, we provide an iterative algorithm for computing the optimal solution to the framework. We also develop an efficient implementation of the algorithm that consists of a series of techniques to make our framework scalable to large data sets. Extensive experimental studies on a paper citation data set and a blog data set demonstrate that our new framework is able to effectively incorporate users' prior knowledge, improves performance over the standard NTF on the task of personalized recommendation, and is scalable to large data sets from real-life applications.

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*

CIKM'10, October 26–29, 2010, Toronto, Ontario, Canada.

General Terms

Algorithms, Experimentation, Measurement, Theory

Keywords

Non-negative Tensor Factorization, Polyadic Data, Prior Knowledge, Iterative Algorithm, Sparse Algorithm

1. INTRODUCTION

Data in many applications are *polyadic*, i.e., they have multiple dimensions. Data in social networks are such an example—for example, data in the blogosphere may have people dimension (the author of a blog post), content dimension (the body of the post), and time dimension (the timestamp of the post). Documents in a digital library are another example—a scientific paper can be described by its authors, keywords, references, publication date, publication venue, etc. To analyze such polyadic data, a very important task is to extract significant characteristics from different data dimensions, where the extracted characteristics can be used either directly for data summarization and visualization or as features for further data analysis. The extracted characteristics can be in the form of, using the blog example, salient communities among bloggers, coherent topics in blog posts, and noteworthy temporal trends of these topics and communities. Because these data dimensions affect each other in a *joint* way, approaches that either analyze each data dimension independently or only consider pairwise relations between data dimensions are not able to accurately capture the data characteristics.

Recently, several multiple-dimensional tensor models have been proposed to capture the higher-order correlation (other than the second order correlation) among various data dimensions. These tensor-based approaches can be categorized into two groups. Approaches in the first group (e.g.,[7, 13]) decompose polyadic data by using higher-order *linear* decompositions, which are extensions of the matrix singular value decomposition. On the other hand, approaches in the second group (e.g., [4, 5, 21]) decompose polyadic data by using non-negative tensor factorizations (NTFs), which are extensions of the non-negative matrix factorization (NMF [18]). The approaches based on NTFs decompose data into *additions* of non-negative components and therefore have many advantages over those based on linear decompositions. Such advantages include ease of interpretation of the extracted characteristics, close connection to the probabilistic models, and no enforcement on the orthogonality among different data characteristics. Because of these advantages, in

this paper we focus on the approaches that extract data characteristics by using NTF.

In an approach based on the standard NTF for extracting data characteristics, the extracted characteristics can be of arbitrary forms and end users do not have any control over them. Such an approach has some benefits—it is simple because it does not require any input other than the observed data. However, such a simple approach also has its weakness: end users have no channel to incorporate their prior knowledge into the process of characteristic extraction. We argue that users' prior knowledge can greatly benefit the extraction of meaningful data characteristics from different perspectives. Cognitively, human concepts such as the content topics usually only occupy a very low dimensional subspace or manifold of the whole space (which usually is of much higher dimensions) and users' prior knowledge can provide guidance toward the appropriate subspace. Statistically, a troublesome issue in the standard NTF-based approach is the problem of overfitting whereas users' input can alleviate this problem. Application-wise, there are many applications in which the end users already have certain domain knowledge, e.g., a pre-constructed ontology of contents, and want to view the data through the lens of such domain knowledge. Therefore, it is beneficial to provide means to allow the end users to incorporate prior knowledge into the process of characteristic extraction.

In this paper, we propose a new framework that extends the standard NTF and allows end users to incorporate prior knowledge in the process of extracting characteristics from polyadic data. We name our framework FacetCube [1]. Our main contributions are summarized as follows.

- We propose a novel non-negative tensor factorization model for extracting data characteristics from polyadic data. In the new model, prior knowledge can be incorporated at three different levels into the factorization and as a result, end users can control the process of characteristic extraction in different data dimensions at different levels. The new model takes the standard NTF as a special case.
- We show that the proposed model has a natural interpretation in a form of the probabilistic generative procedure. From this interpretation, we propose and justify the usage of a Dirichlet prior in the model parameter inference. We further show a technique of controlling the sparseness of the results by using a special Dirichlet prior.
- We develop a very efficient implementation of the algorithm that takes advantage of the sparseness of data, where the implementation has linear (per iteration) time complexity. The implementation contains a series of techniques that are general enough to be equally applicable to any algorithms with similar iterative computations. We further develop an efficient technique of fast computation when the application is to query about the top-K answers.

Our FacetCube framework is general in that it can handle polyadic data of arbitrary order. In this paper, we use two applications, personalized recommendation in a paper citation data set and data exploration in a blog data set, to demonstrate the effectiveness of our framework. Extensive experimental studies on these two data sets demonstrate that our new framework is able to effectively incorporate users' prior knowledge, improves performance over the standard NTF on the task of personalized recommendation, and is scalable to large data sets from real-life applications.

The rest of this paper is organized as the following. In the rest of this section we survey related work. In Section 2 we provide background information. In Section 3 we introduce our FacetCube framework. In Section 4, we offer a probabilistic interpretation of our framework. In Section 5, we describe the techniques for our efficient implementation. We present experimental results in Section 6 and finally give conclusions in Section 7.

Related Work

The majority of existing tensor factorization-based studies focus on *liner* tensor factorizations, where the non-negative constraint is not enforced. Two mostly used linear tensor factorizations are the PARAFAC model [13] and the Tucker model [25], where the latter has recently been generalized by De Lathauwer et al. [7] to the Higher Order Singular Value Decomposition. These models have been used for applications such as Web page ranking [24], cross-lingual information retrieval [3], sensor network analysis [23], etc. We refer interested readers to the comprehensive survey by Kolda and Bader [17]. Although these linear tensor factorizations show some good properties in terms of minimizing certain losses in the Frobenius norm, they also have some weak points. For example, the negative values in the outcome of the factorization make it difficult to interpret the outcome in terms of probabilistic distributions. In addition, in contrast to NTF where the data are factorized into *additions* of components, linear tensor factorizations require *subtractions* among components which make them not intuitively appealing. Furthermore, linear tensor factorizations usually require concepts in the outcome to be orthogonal, and this limits the scope of their applications (e.g., we seldom enforce different temporal trends to be orthogonal to each other). A special linear tensor factorization, the CANDELINC model [2], has certain similarity to our framework in that it allows the data characteristics to be inside a predefined linear subspace. However, for linear factorizations, such a goal can be easily achieved by projecting the data into the subspace before applying the tensor factorization.

Our work is an extension to the existing non-negative tensor factorizations (NTFs) and the following are some related studies in this area. Hofmann [15] proposed the Probabilistic Semantic Analysis (PLSA) for extracting latent topics from documents. Later, Gaussier and Goutte [12] and Ding et al. [9] showed the equivalence between the PLSA and the NMF algorithms by Lee and Seung [18]. Several studies have extended NMFs to NTFs and applied them to applications such as computer vision [22], digital signal processing [11], information retrieval [4], etc. Both Tucker-typed NTF [4, 5, 21] and the PARAFAC-typed NTF [14, 22] have been explored in some of these studies. Our work extends existing NTF approaches by allowing users to incorporate prior knowledge in the factorization process and takes the above standard NTF approaches as special cases. In addition, in this work we focus on the Tucker-typed NTF and the extension to the PARAFAC-typed NTF is straightforward.

Some other related work include the following. Dhillon et

[1] *FacetCube* stands for "factorize data using NTF with coordinates being unconstrained, basis-constrained, or fixed".

al. [8] proposed a co-clustering framework using an information theoretic framework. Long et al. [20] proposed several optimization-based models for multi-type relational data. These approaches, however, focus more on pairwise relations and do not capture higher order correlations. Banerjee et al. [1] proposed a multi-way clustering framework that can handle multi-dimensional relations. However, the approach by Banerjee et al. is derived from the Bregman divergence and it uses an optimization framework. In comparison, our new model has an underlying probabilistic interpretation. In addition, Chi et al. [6] proposed to incorporate certain regularization, such as the connectivity of subgraphs and the smoothness of temporal trends, in the factorization process. Lin et al. [19] proposed a *FacetNet* framework for dynamic community detections. These two studies turned out to solve non-negative *matrix* factorization problems instead of using non-negative *tensor* factorizations.

2. BACKGROUND

2.1 Notations and definitions

First, we introduce some mathematical notations that will be used in this paper. In this paper, unless stated otherwise, values of all variables belong to $\mathcal{R}_+$, the set of non-negative real numbers. We denote scalars by lower-case letters (e.g., a, b, α, β), vectors by lower-case letters in vector forms (e.g., $\vec{p}$, $\vec{q}$), matrices by capital letters (e.g., X, Y, Z), and tensors by calligraphic letters (e.g., $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$). We reserve i, j, k, l, m, n to indicate the indices of tensors, and I, J, K, L, M, N for the sizes of the corresponding dimensions. In the following definitions and the formulations in the rest of the paper, we restrict attention to 3rd-order tensors whereas the same definitions and formulations can be easily generalized to tensors of arbitrary orders.

The *dot product* between tensors $\mathcal{A}$ and $\mathcal{B}$ is denoted by

$$\langle \mathcal{A}, \mathcal{B} \rangle = \sum_{ijk} \mathcal{A}_{ijk} \mathcal{B}_{ijk}$$

Similar dot products are defined on matrices and vectors. For example, $\langle U, V \rangle = tr(U^T V)$, where tr is the trace operator. The *element-wise product* between tensors $\mathcal{A}$ and $\mathcal{B}$ is denoted by

$$\mathcal{C} = \mathcal{A} \circ \mathcal{B}$$

with $\mathcal{C}_{ijk} = \mathcal{A}_{ijk} \mathcal{B}_{ijk}$. Similarly, the *element-wise division* is denoted by

$$\mathcal{C} = \mathcal{A} / \mathcal{B}$$

with $\mathcal{C}_{ijk} = \mathcal{A}_{ijk} / \mathcal{B}_{ijk}$. We define element-wise products and divisions on matrices and vectors in a similar way. The KL-divergence between tensors $\mathcal{A}$ and $\mathcal{B}$ is defined as

$$KL(\mathcal{A} \| \mathcal{B}) = \sum_{ijk} \mathcal{A}_{ijk} \log \mathcal{A}_{ijk} / \mathcal{B}_{ijk} - \sum_{ijk} \mathcal{A}_{ijk} + \sum_{ijk} \mathcal{B}_{ijk}.$$

KL-divergence is defined on matrices and vectors similarly.

We say a tensor $\mathcal{A} \in \mathcal{R}_+^{I \times J \times K}$ is obtained from another tensor $\mathcal{C} \in \mathcal{R}_+^{L \times M \times N}$ by a *base transform*, if we have $\mathcal{A}_{ijk} = \sum_{lmn} X_{il} Y_{jm} Z_{kn} \mathcal{C}_{lmn}$. We denote the base transform by

$$\mathcal{A} = [\mathcal{C}, X, Y, Z].$$

In such a transform, we refer to $\mathcal{C}$ as the *core tensor* and X, Y and Z as the *facet matrices* of the corresponding dimensions. The same notation applies to matrices where $[S, U, V] = USV^T$. In addition, we define shorthand notations for three special base transforms: $\mathcal{C} \times_1 X \doteq [\mathcal{C}, X, I_M, I_N]$, $\mathcal{C} \times_2 Y \doteq [\mathcal{C}, I_L, Y, I_N]$, and $\mathcal{C} \times_3 Z \doteq [\mathcal{C}, I_L, I_M, Z]$, where I_L, I_M, I_N are the identity matrices of sizes L, M, and N, respectively. (In the literature, $\times_1$, $\times_2$ and $\times_3$ are also called the mode-1, mode-2, and mode-3 multiplications of $\mathcal{C}$ [7]).

As the partial derivative of the inner product $\langle \mathcal{A}, [\mathcal{C}, X, Y, Z] \rangle$ with respect to X is independent to X, we denote it by $\langle \mathcal{A}, [\mathcal{C}, \bullet, Y, Z] \rangle$. That is, with $D_{il} = \sum_{jkmn} Y_{jm} Z_{kn} \mathcal{C}_{lmn} \mathcal{A}_{ijk}$, we have

$$\langle \mathcal{A}, [\mathcal{C}, \bullet, Y, Z] \rangle \doteq \frac{\partial}{\partial X} \langle \mathcal{A}, [\mathcal{C}, X, Y, Z] \rangle] = D.$$

In a similar way, we use $\langle \mathcal{A}, [\mathcal{C}, X, \bullet, Z] \rangle$, $\langle \mathcal{A}, [\mathcal{C}, X, Y, \bullet] \rangle$, and $\langle \mathcal{A}, [\bullet, X, Y, Z] \rangle$ to indicate the partial derivatives of $\langle \mathcal{A}, [\mathcal{C}, X, Y, Z] \rangle$ with respect to Y, Z, and $\mathcal{C}$, respectively. Note that we have the following relation

$$\langle \mathcal{A}, [\bullet, X, Y, Z] \rangle \doteq \frac{\partial}{\partial \mathcal{C}} \langle \mathcal{A}, [\mathcal{C}, X, Y, Z] \rangle = [\mathcal{A}, X^T, Y^T, Z^T].$$

2.2 Approach based on the standard NTF

In an NTF-based approach that extracts data characteristics, the first step is to construct a *data tensor*, where the order of the data tensor is the same as the number of dimensions of the data. We again use the blogosphere as an example. A third-order data tensor $\mathcal{A} \in \mathcal{R}_+^{I \times J \times K}$ can be used to represent the blog data where the three dimensions of the data tensor correspond to blogger, keyword, and timestamp, respectively. Each of I, J, and K represents the size of the corresponding data dimensions. I.e., there are in total I bloggers, J keywords and K timestamps in the data. Each entry of the data tensor represents the intensity of the corresponding entry in the observed data. For example, $(\mathcal{A})_{ijk} = a_{ijk}$ can be the frequency that blogger i mentioned keyword j at timestamp k.

Once the data tensor is constructed, in approaches based on the standard NTF, a non-negative tensor factorization is directly applied to the data tensor. The outcomes of this factorization consist of two parts. The first part is a set of *facet matrices* where each matrix represents the most significant characteristics of one dimension of data. More specifically, the number of facet matrices is the same as the order of the tensor and for each facet matrix, each column of the matrix indicates one facet of the corresponding dimension of data. For our blog example, the facet matrices are $X \in \mathcal{R}_+^{I \times L}$, $Y \in \mathcal{R}_+^{J \times M}$, and $Z \in \mathcal{R}_+^{K \times N}$, where each column of X, Y, and Z denotes a salient community of bloggers, a significant topics in posts, and a noteworthy temporal trends, respectively. The second part of the NTF decomposition is a *core tensor* $\mathcal{C}$, which has the same order as the data tensor but usually with a much smaller size. $\mathcal{C}$ represents the correlation among all the facets in all the data dimensions. In our blog example, $\mathcal{C} \in \mathcal{R}_+^{L \times M \times N}$ where L, M, and N are the number of facets in the dimensions of blogger, keyword, and timestamp, respectively.

The target of a non-negative tensor factorization is to find the core tensor $\mathcal{C}$, the facet matrices X, Y, and Z, so that when put together as $[\mathcal{C}, X, Y, Z]$, they approximate $\mathcal{A}$ in an optimal way. A commonly used metric to measure the approximation error is the KL-divergence. That is, the objective of the standard NTF is to find $\mathcal{C}, X, Y$, and Z to

minimize the following error

$$error_{KL} = KL(\mathcal{A}||[\mathcal{C}, X, Y, Z]). \quad (1)$$

In addition, because of the scale-free issue (e.g., $[\mathcal{C}, X, Y, Z] = [\alpha\mathcal{C}, \frac{1}{\alpha}X, Y, Z]$), additional constraints are added so that each column of X, Y, and Z must sum to 1, as well as the sum of all the entries of $\mathcal{C}$ must be 1.

The following procedure, which we proposed in [4, 5], can be used to iteratively searching the optimal solutions for $\mathcal{C}, X, Y$, and Z.

LEMMA 1. *For a given $\mathcal{A} \in \mathcal{R}_+^{I\times J\times K}$, we first define an auxiliary tensor $\mathcal{B} \doteq \mathcal{A}/[\mathcal{C}, X, Y, Z]$. Then the following update rules are guaranteed to converge to an optimal solution to the objective function defined in Equation* (1)*:*

$$\mathcal{C} \leftarrow_0 \mathcal{C} \circ \langle \mathcal{B}, [\bullet, X, Y, Z]\rangle,$$
$$X \leftarrow_1 X \circ \langle \mathcal{B}, [\mathcal{C}, \bullet, Y, Z]\rangle,$$
$$Y \leftarrow_1 Y \circ \langle \mathcal{B}, [\mathcal{C}, X, \bullet, Z]\rangle,$$
$$Z \leftarrow_1 Z \circ \langle \mathcal{B}, [\mathcal{C}, X, Y, \bullet]\rangle.$$

where $\leftarrow_0$ denotes the operation of after all updates are completed, normalizing so that all entries sum to one, and $\leftarrow_1$ denotes the same operation except that all columns are normalized so that they sum to ones.

3. THE FACETCUBE FRAMEWORK

In this section, we describe the details of our FacetCube framework. As we have mentioned in the introduction, incorporating users' prior knowledge can potentially benefit the extraction of data characteristics. The benefits can be of multiple-fold—the extracted characteristics can be more generalizable because of the alleviation of overfitting, they can be more reasonable because they fit users' prior knowledge, and they can meet the users' requirement when the users want to enforce the facets on certain data dimensions. To achieve these benefits, we extend the standard NTF by allowing users' prior knowledge to be incorporated in the process of decomposition. We will discuss two variants in our framework—when the facets in a data dimension are restricted to be in a user-given subspace and when the facets are fixed by the user. Together with the case of unconstrained facets in the standard NTF, our framework allows end users to control the process of data characteristic extraction at three different levels.

3.1 Basis-constrained data dimensions

In the first variant, we assume that the prior knowledge from the end users forms a subspace from which the facets can be located. To illustrate this variant, we first rewrite Eq. (1) as the following equivalent form

$$error_{KL} = KL(\mathcal{A}||[\mathcal{C}, I_I X, I_J Y, I_K Z]), \quad (2)$$

where I_I, I_J, and I_K are identity matrices of dimensions I, J, and K, respectively. We can therefore consider, for example, every facet in Y as a point located in the simplex formed by $\{\vec{e}_1, \ldots, \vec{e}_J\}$ where $\vec{e}_j$ is the unit vector with dimension J (i.e., the j-th element being 1 and other elements being zeros). However, because J can be very large (i.e., tens of thousands of keywords), this simplex can have too large degree of freedoms, and therefore the search space is too large for us to find reliable facets. In such a case, users' prior knowledge can be used to reduce the search space. Assuming the prior knowledge is given as a set of *basis* $Y_B = \{\vec{b}_1, \ldots, \vec{b}_{M'}\}$, then we can reduce the search space of facets to be in the simplex formed by Y_B. I.e., each facet is in the form of $Y_B \cdot \vec{y}$ for certain $\vec{y} \in \mathcal{R}^{M'}$.

To further clarify this, we come back to the blog example. Each facet in the content dimension is a significant topic and in the standard NTF approaches, a topic can be formed by any combination of keywords with arbitrary weights. If the end users provide a set of sub-topics, e.g., obtained from the leaf nodes of a content ontology tree, then our framework can take these sub-topics as a *basis* for the content facets. That is, each facet (topic) must be a convex combination of the given sub-topics. In this case, we say the data dimension is *basis-constrained*. Fig. 1(b) illustrates this point. Another example for basis-constrained data dimension can be the time dimension. Each facet in the time dimension corresponds to a noteworthy temporal trend. According to our prior intuition, a temporal trend should be smooth instead of noise-like. One way to incorporate this prior knowledge is to form a basis consisting of Fourier series in low frequencies. Then a convex combination of this basis will not contain high frequency components and therefore the facets will be guaranteed to be smooth.

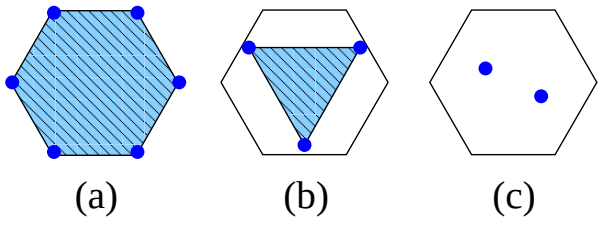

Figure 1: Schematic illustrations of the FacetCube framework with (a) unconstrained, (b) basis-constrained, and (c) fixed data dimensions. In (a) and (b), the shaded areas are the simplex in which the facets can locate; in (c), $\vec{z}_1$ and $\vec{z}_2$ are fixed facets.

In a mathematical form, to handle basis-constrained data dimensions, we extend the objective function in NTF to the following form

$$error_{KL} = KL(\mathcal{A}||[\mathcal{C}, X, Y_B Y, Z]). \quad (3)$$

subject to each column of X, Y, and Z sums to 1. Note that in this objective function, without loss of generality, we assume that Y is the basis-constrained dimension, that Y_B is the basis matrix provided by the end users, and that each column of Y_B sums to 1. As can be observed in Eq. (3), the term $Y_B Y$ replaces the term $I_J Y$ in Eq. (2) and so Eq. (2) can be considered as a special case of Eq. (3).

3.2 Fixed data dimensions

In some other cases, the end users may require the facets for certain data dimensions to be *fixed*. Such a requirement may sound overly restricted, but it is not uncommon in real applications. For example, from the top level nodes of an ontology tree, the user may have already determined the *top-level topics* (in contrast to the *sub-topics* in Sec. 3.1) such as politics, technologies, sports, etc., as well as the representation of these topics as facets. And the user's goal is to summarize the blog posts through the lens of these

pre-constructed facets. As another example, in the time dimension, the user may choose to use a set of facets that correspond to the domain knowledge, i.e., concepts such as year, quarter, month, etc., to detect seasonal trends. In such scenarios, we extend the objective function in NTF to the following form

$$error_{KL} = KL(\mathcal{A}||[\mathcal{C}, X, Y, Z_B]). \quad (4)$$

subject to each column of X, and Y sums to 1. Note that without loss of generality, we assume that the facets in the time dimension are fixed as Z_B and each column of Z_B sums to 1. Because Z_B is fixed, we say that Z is a *fixed data dimension*. Fig 1(c) illustrates this case.

3.3 The overall picture

To summarize the above two variants as well as the third variant which corresponds to the standard NTF case, we give the objective function in its most general form as the following

$$error_{KL} = KL(\mathcal{A}||[\mathcal{C}, X_B X, Y_B Y, Z_B Z]), \quad (5)$$

where X_B, Y_B, and Z_B are given *a priori* whereas X, Y, Z, and $\mathcal{C}$ are to be computed. When X_B, Y_B, and Z_B are set to identity matrices, we obtain the standard NTF problem; when any of the X_B, Y_B, or Z_B is given and the corresponding X, Y, or Z is to be computed, we have the case of the basis-constrained data dimension; when any of the X_B, Y_B, or Z_B is given and the corresponding X, Y, or Z is fixed to be an identity matrix, we have the case of the fixed data dimension.

In addition, because in our proposed algorithm, as will be described momentarily, the (per iteration) optimization of each data dimension is performed independently of other data dimensions, we can adopt any of the above three variants in our FacetCube framework for any data dimensions independently. For example, when using our framework, the end users may decide to let the blogger communities be discovered with no constrains, provide a basis for the contents, and fix the facets in the time dimension.

3.4 An iterative algorithm

We propose an algorithm for computing optimal solutions for our FacetCube framework. We present the algorithm in the most general form—the basis-constrained form—and then show that the unconstrained and the fixed dimensions can be solved by using the same result.

The iterative algorithm and its correctness are stated in the following theorem.

THEOREM 1. *For a given $\mathcal{A} \in \mathcal{R}_+^{I \times J \times K}$, the following update rules are guaranteed to converge to an optimal solution to the objective function defined in Equation* (5).

$$\mathcal{B} \leftarrow \mathcal{A}/[\mathcal{C}, X_B X, Y_B Y, Z_B Z], \quad (6)$$

$$\mathcal{C} \leftarrow_0 \mathcal{C} \circ \langle \mathcal{B}, [\bullet, X_B X, Y_B Y, Z_B Z] \rangle, \quad (7)$$

$$X \leftarrow_1 X \circ \langle \mathcal{B} \times_1 X_B^T, [\mathcal{C}, \bullet, Y_B Y, Z_B Z] \rangle, \quad (8)$$

$$Y \leftarrow_1 Y \circ \langle \mathcal{B} \times_2 Y_B^T, [\mathcal{C}, X_B X, \bullet, Z_B Z] \rangle, \quad (9)$$

$$Z \leftarrow_1 Z \circ \langle \mathcal{B} \times_3 Z_B^T, [\mathcal{C}, X_B X, Y_B Y, \bullet] \rangle. \quad (10)$$

where $\leftarrow_0$ and $\leftarrow_1$ are the same as those defined in Lemma 1.

The proof of this theorem is based on an EM-style argument and is provided in the appendix.

Although the iterative algorithm is given for the case of the basis-constrained dimension, it is straightforward to adopt it to the other two cases. For the case of unconstrained dimension, say $Y_B = I_J$ is the identity matrix, we directly use $\mathcal{B}$ instead of $\mathcal{B} \times_2 Y_B^T$ in Eq. (9) of the update algorithm. For the case of fixed dimension, say $Z_B Z = Z_B$, we simply skip the corresponding update step in Eq. (10). Such a skip is safe because in each step of the iterative algorithm, Eqs. (7)–(10) *independently* improve the objective function in a monotonic way. It is worth noting that another consequence of such independent updates is that the end users can simultaneously enforce different levels of preference at each dimension, e.g., by setting $X_B = I_I$ (unconstrained), providing Y_B (basis-constrained), and fixing $Z_B Z = Z_B$ (fixed) at the same time.

4. A PROBABILISTIC INTERPRETATION

The FacetCube framework turns out to have a natural probabilistic interpretation in that it is equivalent to a special probabilistic generative model. In the following discussion, we focus on the basis-constrained dimension and we use the blog data as an example, assuming the observed data are in a list in the form of $\{\langle \text{blogger } i, \text{word } j, \text{time } k, \mathcal{A}_{ijk} \rangle\}$. We can use the following probabilistic generative procedure to describe how the observed data are sampled:

1. select a community l, a topic m, a temporal trend n, with probability c_{lmn} (this corresponds to an entry of the core tensor $\mathcal{C}$),

2. conditioning on the result in step 1, select a sub-community l', a sub-topic m', a sub-trend n', following $p(l'|l)$, $p(m'|m)$, and $p(n'|n)$ (these correspond to the l-th, m-th, and n-th columns of X, Y, and Z, respectively),

3. conditioning on the results in step 2, select a blogger i, a word j, a time stamp k, following $p(i|l')$, $p(j|m')$, and $p(k|n')$ (these correspond to the l'-th, m'-th, and n'-th columns of X_B, Y_B, and Z_B, respectively).

Then under this generative model, the log-likelihood of the data can be written as

$$\sum_{ijk} \mathcal{A}_{ijk} \log(\sum \mathcal{C}_{lmn} p(l'|l) p(m'|m) p(n'|n) p(i|l') p(j|m') p(k|n')) \quad (11)$$

where the inner sum is computed over all l, m, n, l', m', n'. A simple derivation can show that maximizing the log-likelihood in Eq. (11) is equivalent to minimizing the KL loss in Eq. (5). As a consequence, our FacetCube framework is equivalent to this probabilistic generative model.

The probabilistic interpretation, other than giving additional insights to our FacetCube framework, provides the underpinning for certain extensions of our basic framework. We show such an extension which is to consider a maximum a posteriori (MAP) estimation for the basic probabilistic model. Here, we consider a case of using the Dirichlet distribution as the prior distribution of the parameters. First, we consider the prior for X. The prior of each column of X is a Dirichlet distribution with hyper-parameter $\alpha_X > 0$. The logarithm of the prior probability is

$$\ln P(X) = (\alpha_X - 1) \sum_{il} \ln X_{il} + c_X,$$

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 error for the MAP estimation is that for the MLE plus the logarithm of the prior probabilities:

$$error_{MAP} \doteq error_{KL} - \ln P(X) - \ln P(Y) - \ln P(Z) - \ln P(\mathcal{C}). \quad (12)$$

With a few derivations, we can obtain the following algorithm for solving the MAP estimation. (Due to the limit of the space, we skip the details of the proof).

COROLLARY 1. *The following update rules are guaranteed to converge to an optimal solution to the objective function defined in Equation* (12).

$$\mathcal{B} \leftarrow \mathcal{A}/[\mathcal{C}, X_B X, Y_B Y, Z_B Z],$$
$$\mathcal{C} \leftarrow_0 [\mathcal{C} \circ \langle \mathcal{B}, [\bullet, X_B X, Y_B Y, Z_B Z]\rangle + (\alpha_{\mathcal{C}} - 1)]_\varepsilon,$$
$$X \leftarrow_1 \left[X \circ \langle \mathcal{B} \times_1 X_B^T, [\mathcal{C}, \bullet, Y_B Y, Z_B Z]\rangle + (\alpha_X - 1)\right]_\varepsilon,$$
$$Y \leftarrow_1 \left[Y \circ \langle \mathcal{B} \times_2 Y_B^T, [\mathcal{C}, X_B X, \bullet, Z_B Z]\rangle + (\alpha_Y - 1)\right]_\varepsilon,$$
$$Z \leftarrow_1 \left[Z \circ \langle \mathcal{B} \times_3 Z_B^T, [\mathcal{C}, X_B X, Y_B Y, \bullet]\rangle + (\alpha_Z - 1)\right]_\varepsilon.$$

where ε is a small positive real number and $[\cdot]_\varepsilon$ stands for taking the maximal one between the variable value and ε.

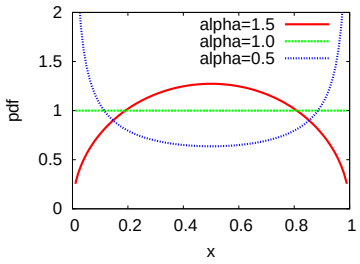

Figure 2: Distribution of variables with different α. When $\alpha > 1$, the mode is at 0.5, and so x tends to be non-zeros. When $\alpha < 1$, the modes are 0 and 1, which make x or $1 - x$ tend to be 0, an so it results in sparse solutions.

Corollary 1 reveals that the Dirichlet prior can play two different roles in computing the optimal solutions. When $\alpha > 1$, the Dirichlet prior plays a role of *smoothing* by adding pseudo-counts to each variable. On the other hand, when $\alpha < 1$, the Dirichlet prior plays a role opposite to smoothing—it pulls variable with small values into the negative territory (which are then set to ε by the update rules) and as a result, making the solution more *sparse* (if we disregard the small-valued ε's). This point is demonstrated in Fig. 2 by using a two-variable Dirichlet (Beta) distribution.

5. IMPLEMENTATION OF ALGORITHM

In this section, we first describe some techniques we developed to make our algorithm practical, and then introduce a technique we adopted for fast computation of top-K queries. The reason for us to give such a detailed description is that we believe these techniques are important by themselves: they can be equally applied to other multiplicative update rules for tensors of arbitrary order and they can be used to solve a family of similar problems.

5.1 Implementation details

A naive implementation of the update rules in Theorem 1 is impractical for all but very small data sets. In real applications, the size of a data dimension (e.g., the number of bloggers or the size of the vocabulary) can easily be tens of thousands. When $I = J = K = 10,000$, for example, even the computation of $[\mathcal{C}, X, Y, Z]$ in Eq. (6) is unmanageable, for $[\mathcal{C}, X, Y, Z]$ is a dense tensor with 1 trillion entries.[2] To handle data from real applications, we designed an efficient implementation of the FacetCube algorithm that exploits three key insights we observed. Part of these insights have been roughly mentioned in a high-level way in [4, 5]. Here we give them a much more rigorous and complete treatment.

In the following discussion, without loss of generality, it is assumed that $L \geq M \geq N$ and $I \geq J \geq K$. In addition, we assume that each data record 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 a blogger mentions a keyword on a particular day. It is also assumed that the data records have been sorted according to $key3$ as the major key and then $key2$ and then $key1$ as the minor keys. We shall come back to this last assumption later.

The first insight we have exploited is that data in real applications are usually sparse. For example, not every blogger uses every word in every of his or her posts. Because $\mathcal{A}$ is a sparse tensor, not every entry of $[\mathcal{C}, X, Y, Z]$ is needed. In our implementation, we take advantage of data sparseness by computing base transforms in an *on-demand* fashion. For example, it only takes time $O(n_z \cdot L^3)$ to compute the entries in $[\mathcal{C}, X, Y, Z]$ that correspond to the non-zero entries in $\mathcal{A}$, where n_z is the total number of non-zero entries in $\mathcal{A}$.

The second insight is that the update rules in Theorem 1 involve a lot of nested summations and multiplications that have a lot of structures. Therefore if we carefully order these nested operations and cache the intermediate results, we can avoid generating them multiple times. More specifically, we rewrite Eq. (6) as the following by pushing certain summations inside the nest

$$\mathcal{B}_{ijk} = \mathcal{A}_{ijk} / \sum_l X_{il} (\sum_m Y_{jm} (\sum_n Z_{kn} \mathcal{C}_{lmn})). \quad (13)$$

In the above expression, for entries with the same k index, the term in the inner parentheses can be reused even when the i and j indices vary; similarly, for entries with the same j and k indices, the term in the outer parentheses can be reused even when the i index varies. We can derive similar expressions for Eqs. (7)–(10). For instance, Eq. (7) can be rewritten as

$$[\mathcal{B}, X^T, Y^T, Z^T]_{lmn} = \sum_k Z_{kn} (\sum_j Y_{jm} (\sum_i X_{il} \mathcal{B}_{ijk})). \quad (14)$$

And for Eq. (8), we first notice that

$$\langle \mathcal{B} \times_1 X_B^T, [\mathcal{C}, \bullet, Y, Z]\rangle = X_B^T \langle \mathcal{B}, [\mathcal{C}, \bullet, Y, Z]\rangle$$

[2]For the discussion in this part we restrict attention to the case of unconstrained dimensions, for the computations for $X_B X$, $Y_B Y$, and $Z_B Z$ do not dominate the time complexity.

and for the second part of the above expression, we have

$$\langle \mathcal{B}, [\mathcal{C}, \bullet, Y, Z] \rangle_{il} = \sum_{jk} \mathcal{B}_{ijk} (\sum_{m} Y_{jm} (\sum_{n} Z_{kn} \mathcal{C}_{lmn})) \quad (15)$$

As a result, we can reuse intermediate computation results in all the update rules, since $\mathcal{A}$ is stored with $key3$ (the k index) as the major key and then $key2$ (the j index) and then $key1$ (the i index) as the minor keys. The sorting of $\mathcal{A}$ does affect the time complexity for two reasons. On the one hand, the indices i, j, and k are positive integers with known upper-bounds and so a linear sorting algorithm, such as bucket sort, can be applied. On the other hand, $\mathcal{A}$ only has to be sorted once before the iterative process starts and so the sorting cost is amortized among multiple iterations. It is worth mentioning that $\mathcal{B}$, which is also sparse, does not have to be explicitly sorted—it is automatically sorted because of the way it is derived from the sparse tensor $\mathcal{A}$.

The third insight is that different data dimensions usually have different cardinalities. For example, if the time unit is a day, then the size of the time dimension can be much smaller than than of the blogger dimension. We take advantage of this fact by ordering the computation in such a way that the dimension of the smallest size is put in the inner parentheses. A moment of thoughts on Eqs. (13)–(15) can show that this ordering is always possible as long as $\mathcal{A}$ is sorted accordingly at the beginning. As will be demonstrated in the experimental studies, this re-ordering makes huge difference in the running time of our algorithm.

Algorithm for updating X

▷ input: $\mathcal{B}$ as $\{\langle key1, key2, key3, v\rangle\}$,$X$,$Y$,$Z$,$\mathcal{C}$,$X_B$
▷ output: updated X
1: $k \leftarrow -1$, $j \leftarrow -1$, $i \leftarrow -1$, $E \leftarrow \mathbf{0}$;
2: **for each** entry $\langle key1, key2, key3, v\rangle$ of $\mathcal{B}$ **do**
3: **if** $k \neq key3$
4: $k \leftarrow key3$, $j \leftarrow -1$;
5: construct D s.t. $D_{lm} \leftarrow \sum_n Z_{kn} \mathcal{C}_{lmn}$;
6: **if** $j \neq key2$
7: $j \leftarrow key2$;
8: construct $\vec{d}$ s.t. $(\vec{d})_l \leftarrow \sum_m Y_{jm} D_{lm}$;
9: $i \leftarrow key1$;
10: $E_{row_i} \leftarrow E_{row_i} + v \cdot \vec{d}^T$;
11: $X \leftarrow_1 X \circ (X_B^T E)$;
12: **return** X;

Figure 3: Implementation of the update rule for X. Those for Y, Z, and $\mathcal{C}$ are similar.

To summarize, as an example, Fig. 3 gives the high-level pseudo code for our implementation of the update rule for updating X, for the case of basis-constrained dimension. The time complexity can be shown to be $O(n_z \cdot L + n_p \cdot L^2 + K \cdot L^3)$, where n_z is the number of non-zero entries in $\mathcal{A}$, n_p is the number of distinct (j, k) pairs in $\mathcal{A}$, and K is the smallest size among all the data dimensions. As can be observed from the time complexity, if we consider the number of factors L as a constant, then the per iteration time is linear in the size of raw data n_z; if on the other hand we do not consider L to be a constant, because $n_z > n_p > K$, we assign the lowest coefficient to the most expensive L^3 term and therefore fully exploit the data characteristics. We will verify these analyses in the experimental studies.

5.2 Fast computation for top-K queries

In many recommendation applications, instead of being interested in the ranks of the whole list of candidates, the end users often are only interested in a quick answer to the top-K queries. For example, the query may be "Who are the top 3 bloggers mostly involved in a topic during a given time period?" or "What are the top 10 references that are mostly relevant to a group of authors who plan to co-author a paper on a set of keywords?". Because in many applications such queries must be answered in real time, fast computation of top-K answers becomes crucial. In the implementation of our framework, we develop a technique that derives the *exact* top-K answers without computing the scores of all the candidates for the recommendation. Without going into details, we want to point out that the key component of the technique is the Fagin's algorithm [10] which has been extensively used in the database field for answering fuzzy queries over multimedia databases. For Fagin's algorithm to work, a key requirement is that the score function must be *monotonic*. A function $f(z_1, \ldots, z_N)$ is monotonic if $\{z'_1 \geq z_1, \ldots, z'_N \geq z_N\}$ implies that $f(z'_1, \ldots, z'_N) \geq f(z_1, \ldots, z_N)$. It turns out that our ranking function satisfies this condition of monotonicity. We illustrate this by using the above query of recommending references to a given set of authors on a given set of keywords. We assume that X, Y, and Z correspond to *author*, *keyword*, and *reference*, and X_B, Y_B, and Z_B are the corresponding basis.

Recall that $\mathcal{A} \sim [\mathcal{C}, X_B X, Y_B Y, Z_B Z]$ and so for a set of authors and a set of keywords, the relevances of the references are $[\mathcal{C}, \vec{x}^T, \vec{y}^T, Z_B Z] = (Z_B Z) \cdot [\mathcal{C}, \vec{x}^T, \vec{y}^T, I_N]$, where the k-th row of the result indicates the relevance of the k-th reference. Because $\vec{x}^T$ and $\vec{y}^T$ are obtained by aggregating the given set of authors and the given set of keywords, respectively, entries in $\vec{c} = [\mathcal{C}, \vec{x}^T, \vec{y}^T, I_N]$ are non-negative. Because $Z_B Z$ is also non-negative, it can be easily shown that the score function $f[(Z_B Z)_{row_k}] = (Z_B Z)_{row_k} \cdot \vec{c}$ is a monotonic function.

6. EXPERIMENTAL STUDIES

In this section, we present experimental studies on two data sets—a paper citation data set and a blog data set. For experiments on the paper citation data, we study the task of personalized reference recommendation, and we focus on the accuracy of recommendations, the running time of our factorization algorithm, the time for computing the top-K recommendations, as well as the impact of the Dirichlet prior. For experiments on the blog data, we illustrate a real application scenario where the end users have an ontology as the facets for content, and a fixed set of facets for time.

6.1 The CiteSeer Data

The paper citation data set is obtained from the CiteSeer website[3]. This data set contains the information about papers published in the computer science area. The information we used includes the authors, the abstract, and the citations of each paper. The abstracts of the papers are pre-processed through the following steps. First, an NLP package[4] is used to detect and only keep the nouns from the abstracts (according to our experiences, non-noun words are not very informative); then a standard set of stop words are

[3]http://citeseer.ist.psu.edu/
[4]http://opennlp.sourceforge.net/

Table 1: Performance of different algorithms on the task of recommending references on CiteSeer data

	top-1	top-5	top-10	top-50	top-100
baseline	**0.113**	0.316	0.438	0.810	1.007
NTF	0.109	0.322	0.456	0.864	1.081
FacetCube	0.110	**0.326**	**0.463**	**0.870**	**1.088**
FacetCube-D	0.110	0.325	0.462	0.868	1.086

removed; finally, the Porter stemming algorithm is applied to normalize the keywords. After these steps, we use certain thresholds to filter out uncommon words, authors with few publications, and references being cited too infrequently. The final data set contains 15,072 papers with 6,821 distinct authors, 1,790 distinct keywords, and 21,894 distinct references.

We design the task as for a set of authors and a set of keywords, to recommend the most relevant references. Such recommendations can be helpful for authors to select a set of candidate references when writing a paper on a particular topic. The performance is measured in the following way. The 15,072 papers are randomly split evenly into a training set and a testing set. Papers in the training set are used to construct the data tensor where the three data dimensions are: *author*(X), *keyword*(Y), and *reference*(Z). The standard NTF and FacetCube are applied to the data tensor to compute the facet matrices and the core tensor. During testing, for each paper in the testing set, we assume its authors and keywords are given. Relevant references are to be recommended by using $[\mathcal{C}, \vec{x}^T, \vec{y}^T, Z] = Z \cdot [\mathcal{C}, \vec{x}^T, \vec{y}^T, I_N]$, where $\vec{x}^T$ and $\vec{y}^T$ are obtained by aggregating the authors and keywords, respectively, of the given paper. The performance is measured by comparing the recommendations with the ground truth, namely the real references cited in the papers in the testing set. The metric we used for the measurement is the average Normalized Discounted Cumulative Gain (NDCG [16]) for the top-K answers, where NDCG is defined as

$$NDCG = \sum_{k=1}^{K} \frac{2^{rel(k)} - 1}{\log(1+k)}.$$

In our performance study, we choose to use a binary relevance score for $rel(k)$, i.e., $rel(k) = 1$ if item k occurs in the testing data and 0 otherwise.

For the FacetCube algorithm, we show a case where the prior knowledge is used to improve the personalized recommendation. We construct a basis X_B for the authors by using the co-authorship relationship. More specially, we set X_B to be the normalized version of $I + \alpha W$ where I is the identity matrix and W is the adjacency matrix representing the co-authorship relations in the training data, i.e., $W_{ij} = 1$ if authors i and j have co-authored a paper and 0 otherwise. X_B can actually be considered as the first order approximation to the diffusion kernel $(I - \alpha W)^{-1}$ and therefore by using X_B, we restrict the facets of authors to be embedded in the manifold of this diffusion kernel. In our experiments, we simply set $\alpha = 0.01$.

We compare FacetCube with (a) the standard NTF and (b) a baseline in which the counts of reference for each keyword in the training data are stored and used to rank the references in the testing data. The baseline gives global (non-personalized) recommendations. For both the standard NTF and FacetCube, we set the facet numbers to be $L = M = N = 50$. Table 1 reports the NDCG scores at top-K, for $K = 1, 5, 10, 50, 100$. As can be seen from the results, FacetCube slightly outperforms the standard NTF in all the cases and outperforms the baseline in most of the cases. FacetCube improves over the standard NTF because intuitively, we enforce the authors who have co-authored papers in the training set to be in similar facets in the factorization.

To investigate the running time of our implementation, we conduct the following experiments. First, we construct a series of data sets with different sizes by randomly removing a portion of references. Fig. 4(a) shows the running time of FacetCube under different data sizes. For the factorizations, we set the condition for convergence as the relative error being less than 10^{-4}, for which all the methods converge within a couple hundreds of iterations. The results in Fig. 4(a) verify that the running time of our implementation is linear in the size of data set. Second, we fix the data set and test a wide range of facet numbers, from 10 to 100. In Fig. 4(b) we show the running time under different facet numbers for two ways of ordering data: one by $\langle reference, author, keyword \rangle$ (best) and anther by $\langle keyword, author, reference \rangle$ (worst). The former is the best because in this data set, the number of distinct keywords is much smaller than that of the references. From the curves we can see that the running time is about $O(L^3)$ (note the log-log scale in the figure). Also revealed is that a good order can make the implementation 10 to 50 times faster for this specific data set.

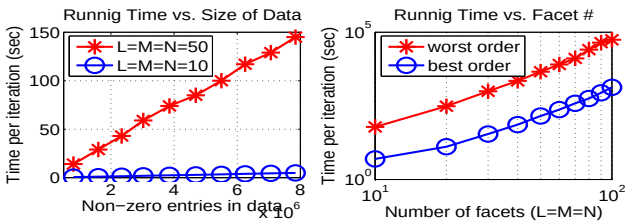

Figure 4: Running time vs. number of non-zero entries in the data tensor (a), and vs. the number of facets (b).

In Fig. 5(a), we report the time for computing the top-K queries by our implementation. Also shown is the lower bound of a naive implementation that computes the scores for all the candidates (for the naive approach, we only measure the time for computing the scores but not for ranking, and so it is a lower bound). As can be seen, our computation is much faster. For example, for the top-1 query, a naive approach will take at least 75 seconds for the testing data while our implementation takes less than 1 second. Next, we study how the Dirichlet prior influences the sparseness of the results. We run FacetCube with $\varepsilon = 10^{-100}$ and $\alpha_Z - 1$ to be a wide range of values. We report in Fig. 5(b) the number of entries in the facet matrix Z that are smaller than or equal to ε. As can be seen, with smaller values of $\alpha_Z - 1$, we makes the facets more sparse. However, this sparseness has its costs. In Table 1, on the row of FacetCube-D, we give the NDCG scores for FacetCube with $\alpha_Z - 1$ set to be -10^{-50}. As can be seen, the performance deteriorates a little bit.

6.2 The Blog Data

The blog data was collected by an NEC in-house blog crawler. This data set contains 1,161 bloggers, 280,259 blog

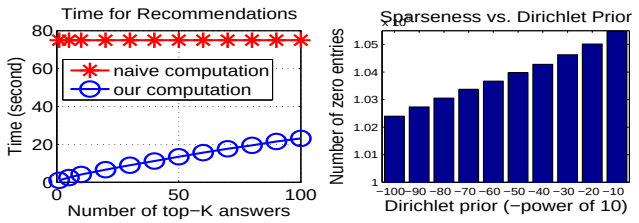

Figure 5: Time for computing the top-K answers (a), and the level of sparsity (b).

posts, with 19,390 keywords (nouns only), for 28 consecutive weeks, between March 02 to September 09, 2008. In addition, we constructed an ontology for content by using data from the Open Directory Project (http://www.dmoz.org/). The ontology consists of 296 tree nodes, such as "Computers_Hardware" and "Society_Politics". For each of the tree node, a description vector is constructed by aggregating the descriptions of all the Web sites under the given tree node.

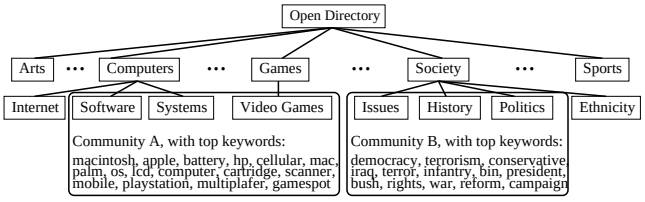

Figure 6: Two representative communities and their projections on the Open Directory ontology.

In this experiment we demonstrate a scenario where the user's domain knowledge is used to directly control the factorization in the FacetCube framework. For this purpose, we fix the facets in the content dimension to be the 296 description vectors and the facets in the time dimension to be 28 windowing functions that correspond to the 28 weeks in our data. Therefore, the free variables in this experiment are the blogger facet matrix X and the core tensor $\mathcal{C}$. We set the number of blogger facets (i.e., the number of blogger communities) to be 10.

It turns out that the blogger communities derived by the FacetCube algorithm all have certain focused topics, such as technology, politics, music, etc. In Fig. 6 we illustrate two representative blogger communities and their projections on the Open Directory ontology. The projections are computed by aggregating out the time dimension of the core tensor $\mathcal{C}$ and from the resulting matrix, identifying for each community the top 3 ontology tree nodes that have the largest values. As can be seen, community A focuses on different issues related to technology while community B focuses on political issues (as verified by the top keywords among the top bloggers in the communities).

Finally in Fig. 7, we show the temporal trends for these two communities by aggregating out the content dimension from the core tensor $\mathcal{C}$. In addition, we also show some hot keywords among the community members during certain temporal peaks. As can be seen, these hot keywords clearly indicate certain noteworthy events (e.g., the release of 3G iPhone in community A and the nomination of the vice presidential candidates in community B).

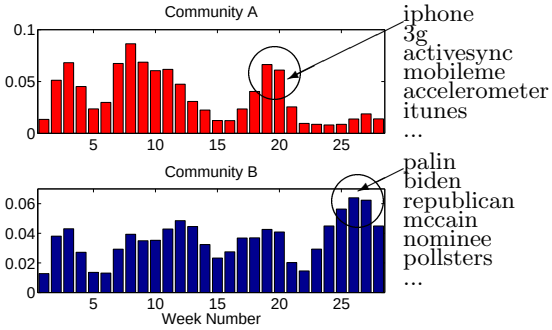

Figure 7: Temporal trends for the two representative communities.

7. CONCLUSIONS

In this paper, we presented a novel framework, FacetCube, that extends the standard non-negative tensor factorization for extracting data characteristics from polyadic data. The FacetCube framework allows end users great flexibility in incorporating their prior knowledge at different levels in the process of characteristic extraction. In addition to describing the new framework from the NTF point of view, we also made connection to the perspective of probabilistic generative models. Furthermore, we provided iterative algorithms for solving the corresponding optimization problems as well as efficient implementations for handling large-scale data from real-life applications. Extensive experimental studies demonstrated the effectiveness of our new framework as well as the efficiency of our algorithm.

Acknowledgments

The authors would like to thank Professor C. Lee Giles for providing the CiteSeer data set, and thank Koji Hino and Junichi Tatemura for helping prepare the blog data set.

8. REFERENCES

[1] A. Banerjee, S. Basu, and S. Merugu. Multi-way clustering on relation graphs. In *SIAM Int. Conf. on Data Mining*, 2007.

[2] J. D. Carroll, S. Pruzansky, and J. B. Kruskal. CANDELINC: A general approach to multi-dimensional analysis of many-way arrays with linear constraints on parameters. *Psychometrika*, 45, 1980.

[3] P. A. Chew, B. W. Bader, T. G. Kolda, and A. Abdelali. Cross-language information retrieval using PARAFAC2. In *Proc. of the 13th SIGKDD Conference*, 2007.

[4] Y. Chi, S. Zhu, Y. Gong, and Y. Zhang. Probabilistic polyadic factorization and its application to personalized recommendation. In *Proc. of the 17th CIKM Conference*, 2008.

[5] Y. Chi, S. Zhu, K. Hino, Y. Gong, and Y. Zhang. iOLAP: A framework for analyzing the internet, social networks, and other networked data. *Multimedia, IEEE Transactions on*, 11(3):372–382, April 2009.

[6] Y. Chi, S. Zhu, X. Song, J. Tatemura, and B. L. Tseng. Structural and temporal analysis of the blogosphere through community factorization. In *Proc. of the 13th SIGKDD Conference*, 2007.

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

[8] I. S. Dhillon, S. Mallela, and D. S. Modha. Information-theoretic co-clustering. In *Proc. of the 9th ACM SIGKDD Conference*, 2003.

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

[10] R. Fagin. Fuzzy queries in multimedia database systems. In *Proc. of the 17th ACM Symposium on Principles of Database Systems*, 1998.

[11] D. FitzGerald, M. Cranitch, and E. Coyle. Non-negative tensor factorisation for sound source separation. In *Proc. of the Irish Signals and Systems Conference*, 2005.

[12] E. Gaussier and C. Goutte. Relation between plsa and nmf and implications. In *Proc. of the 28th SIGIR Conference*, 2005.

[13] 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.

[14] T. Hazan, S. Polak, and A. Shashua. Sparse image coding using a 3d non-negative tensor factorization. In *Proc. of the 10th ICCV Conference*, 2005.

[15] T. Hofmann. Unsupervised learning by probabilistic latent semantic analysis. *Mach. Learn.*, 42(1-2), 2001.

[16] K. Järvelin and J. Kekäläinen. IR evaluation methods for retrieving highly relevant documents. In *Proc. of the 23rd SIGIR conference*, 2000.

[17] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Review*. to appear.

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

[19] Y. Lin, Y. Chi, S. Zhu, H. Sundaram, and B. L. Tseng. FacetNet: A framework for analyzing communities and their evolutions in dynamic networks. In *Proc. of the 17th WWW Conference*, 2008.

[20] B. Long, Z. Zhang, and P. S. Yu. A probabilistic framework for relational clustering. In *Proc. of the 13th SIGKDD Conference*, 2007.

[21] M. Mørup, L. K. Hansen, and S. M. Arnfred. Algorithms for sparse nonnegative Tucker decompositions. *Neural Comput.*, 20(8):2112–2131, 2008.

[22] A. Shashua and T. Hazan. Non-negative tensor factorization with applications to statistics and computer vision. In *Proc. of the 22nd ICML Conference*, 2005.

[23] J. Sun, C. Faloutsos, S. Papadimitriou, and P. S. Yu. GraphScope: parameter-free mining of large time-evolving graphs. In *Proc. of the 13th SIGKDD Conference*, 2007.

[24] J.-T. Sun, H.-J. Zeng, H. Liu, Y. Lu, and Z. Chen. CubeSVD: a novel approach to personalized web search. In *Proc. of the 14th WWW Conference*, 2005.

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

Appendix: Proof for Theorem 1

PROOF. Assume that the values obtained from the previous iteration are $\tilde{X}$, $\tilde{Y}$, $\tilde{Z}$, and $\tilde{\mathcal{C}}$, respectively. We prove the update rule for X. The rules for Y, Z, and $\mathcal{C}$ can be proved similarly. For the update rule of X, we can consider Y, Z, and $\mathcal{C}$ as fixed (i.e., fixed as their values $\tilde{Y}$, $\tilde{Z}$, and $\tilde{\mathcal{C}}$ in the previous iteration). To avoid notation clutters, we define $\tilde{\tilde{X}} \doteq X_B\tilde{X}$, $\tilde{\tilde{Y}} \doteq Y_B\tilde{Y}$, $\tilde{\tilde{Z}} = Z_B\tilde{Z}$, and we rewrite the objective function as

$$\min D_X(X) = \min KL(\mathcal{A}||[\tilde{\mathcal{C}}, X_B X, \tilde{\tilde{Y}}, \tilde{\tilde{Z}}])$$

First define

$$\gamma_{ijklmnl'} = \tilde{\mathcal{C}}_{lmn}(X_B)_{il'}\tilde{X}_{l'l}\tilde{\tilde{Y}}_{jm}\tilde{\tilde{Z}}_{kn}$$

and

$$\theta_{ijklmnl'} = \frac{\tilde{\mathcal{C}}_{lmn}(X_B)_{il'}\tilde{X}_{l'l}\tilde{\tilde{Y}}_{jm}\tilde{\tilde{Z}}_{kn}}{[\tilde{\mathcal{C}}, \tilde{\tilde{X}}, \tilde{\tilde{Y}}, \tilde{\tilde{Z}}]_{ijk}} = \frac{\gamma_{ijklmnl'}}{[\tilde{\mathcal{C}}, \tilde{\tilde{X}}, \tilde{\tilde{Y}}, \tilde{\tilde{Z}}]_{ijk}}$$

where obviously we have $\sum_{ijklmnl'} \theta_{ijklmnl'} = 1$.

Then we have

$$\begin{aligned}
&D_X(X)\\
=&\sum_{ijk}\left[\sum_{lmnl'} \tilde{\mathcal{C}}_{lmn}(X_B)_{il'}X_{l'l}\tilde{\tilde{Y}}_{jm}\tilde{\tilde{Z}}_{kn}\right.\\
&\left.-\mathcal{A}_{ijk}\ln\left(\sum_{lmnl'}\tilde{\mathcal{C}}_{lmn}(X_B)_{il'}X_{l'l}\tilde{\tilde{Y}}_{jm}\tilde{\tilde{Z}}_{kn}\right)\right] + c_1\\
\le&\sum_{ijklmnl'}\left[\tilde{\mathcal{C}}_{lmn}(X_B)_{il'}X_{l'l}\tilde{\tilde{Y}}_{jm}\tilde{\tilde{Z}}_{kn}\right.\\
&\left.-\mathcal{A}_{ijk}\theta_{ijklmnl'}\ln\frac{\sum_{lmnl'}\tilde{\mathcal{C}}_{lmn}(X_B)_{il'}X_{l'l}\tilde{\tilde{Y}}_{jm}\tilde{\tilde{Z}}_{kn}}{\theta_{ijklmnl'}}\right] + c_1\\
=&\sum_{ijklmnl'}\left[\tilde{\mathcal{C}}_{lmn}(X_B)_{il'}X_{l'l}\tilde{\tilde{Y}}_{jm}\tilde{\tilde{Z}}_{kn}\right.\\
&\left.-\gamma_{ijklmnl'}\tilde{\mathcal{B}}_{ijk}\ln\left(\tilde{\mathcal{C}}_{lmn}(X_B)_{il'}X_{l'l}\tilde{\tilde{Y}}_{jm}\tilde{\tilde{Z}}_{kn}\right)\right] + c_2\\
=&-\sum_{ijklmnl'}\left[\gamma_{ijklmnl'}\tilde{\mathcal{B}}_{ijk}\ln\left(\tilde{\mathcal{C}}_{lmn}(X_B)_{il'}X_{l'l}\tilde{\tilde{Y}}_{jm}\tilde{\tilde{Z}}_{kn}\right)\right] + c_3\\
\doteq&Q(X;\tilde{X}),
\end{aligned}$$

where c_1, c_2, and c_3 are constants irrelevant to X. Note that in the last step of the above derivation, we used the fact that $\sum_{ijklmnl'}\tilde{\mathcal{C}}_{lmn}(X_B)_{il'}X_{l'l}\tilde{\tilde{Y}}_{jm}\tilde{\tilde{Z}}_{kn} = \sum_{ijk}\tilde{\mathcal{C}}_{lmn}$ because the columns of X all sum to 1.

It can be easily shown that $Q_X(X;\tilde{X})$ is an *auxiliary function* of $D_X(X)$ in the sense that

1. $D_X(X) \le Q_X(X;\tilde{X})$, 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

$$L(X,\vec{\lambda}) = Q(X;\tilde{X}) + \vec{\lambda}^T(X^T\vec{1}_L - \vec{1}_{L'}),$$

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

$$\frac{\partial L}{\partial X_{l'l}} = \frac{\tilde{X}_{l'l}}{X_{l'l}}\sum_{ijkmn}\tilde{\mathcal{B}}_{ijk}\tilde{\mathcal{C}}_{lmn}\tilde{\tilde{Y}}_{jm}\tilde{\tilde{Z}}_{kn} + \lambda_l = 0$$

$$\frac{\partial L}{\partial \lambda_l} = \sum_{l'}X_{l'l} - 1 = 0$$

which gives the update rule for X in Theorem 1. □