HKU CS Tech Report TR-2010-10

CubeLSI: An Effective and Efficient Method for Searching Resources in Social Tagging Systems

Bin Bi, Sau Dan Lee, Ben Kao, Reynold Cheng
{bbi, sdlee, kao, ckcheng}@cs.hku.hk

Technical Report TR-2010-10
Department of Computer Science
The University of Hong Kong

2010-11-16

Abstract

In a social tagging system, resources (such as photos, video and web pages) are associated with tags. These tags allow the resources to be effectively searched through tag-based keyword matching using traditional IR techniques. We note that in many such systems, tags of a resource are often assigned by a diverse audience of causal users (taggers). This leads to two issues that gravely affect the effectiveness of resource retrieval: (1) Noise: tags are picked from an uncontrolled vocabulary and are assigned by untrained taggers. The tags are thus noisy features in resource retrieval. (2) A multitude of aspects: different taggers focus on different aspects of a resource. Representing a resource using a flattened bag of tags ignores this important diversity of taggers. To improve the effectiveness of resource retrieval in social tagging systems, we propose CubeLSI — a technique that extends traditional LSI to include taggers as another dimension of feature space of resources. We compare CubeLSI against a number of other tag-based retrieval models and show that CubeLSI significantly outperforms the other models in terms of retrieval accuracy. We also prove two interesting theorems that allow CubeLSI to be very efficiently computed despite the much enlarged feature space it employs.

1 Introduction

We are witnessing an increasing number of social tagging systems on the web, such as *Flickr*, *Delicious*, *Bibsonomy*, *Last.fm* and *YouTube*. These systems provide platforms on which various *resources*, such as photos, music, video and web sites, are shared. One of the distinctive characteristics of these social tagging systems is that resources are tagged by communities of users. Through manually-assigned tags, users can express their distinct interests on the various aspects of the resources. For example, a photo could be tagged with words that describe the subjects, people and places in the photo; the artistic elements of the photo; or the historical background of the photo. Given the collective wisdom of a user community, tagging amasses a tremendous amount of meta-data, which facilitates the organization and retrieval of resources. As an example, a project called "The Commons"[1] was launched by *Flickr* in 2008 in which photography collections obtained from museums and libraries are displayed. One of the objectives of "The Commons" is to provide a way for the general public to contribute information and knowledge about the photos by means of written comments and tags. With their increasing popularity, social tagging systems have been growing at very high rates, both in terms of their user communities and resource collections. For example, *Flickr* had more than 32 million users as of May 2009[2] and hosted over 4 billion photos as of October 2009[3]. *Last.fm* had 30 million users and

[1] http://www.flickr.com/commons

[2] http://www.flickr.com/help/forum/en-us/97258/

[3] http://blog.flickr.net/en/2009/10/12/4000000000/

more than 7 million tracks cataloged as of March 2009[4]. The sheer amount of resources made available by these systems calls for highly effective and scalable searching methods. Our goal is to study the properties of social tagging systems and to propose a novel tag-based technique to search for relevant resources.

Our search technique, called CubeLSI, follows a typical keyword-based query model employed by major social tagging systems. In these systems, a user query is expressed by a few keywords (tags) and a ranked list of relevant resources is returned as the query result. A simple approach to this searching problem is to match the query keywords against the tags associated with each resource in a way that is similar to traditional document retrieval techniques. This approach, however, suffers from two complications that are intrinsic to social tagging systems. Firstly, taggers are typically untrained casual users who are free to pick their own tags. The tags thus come from an uncontrolled vocabulary rather than from a well-defined taxonomy. This results in ambiguities: a single tag could refer to multiple different concepts (polysemy) and a single concept could be described by different tags (synonymy). These make tags a noisy feature of resources in the retrieval process. Secondly, while a traditional document is usually written by a single author, a resource is typically tagged by a diverse audience of taggers. Different interest groups of taggers may focus on different aspects of the same resource. For example, a group of taggers may be interested in the photo-taking techniques (e.g., composition, color-tone, exposure, etc.) of a photo, while another group may be interested in its content (e.g., faces and places). An interesting implication to this observation is that not only "*which* tags are assigned to a resource" is important information, but also "*who* has assigned the tags," which provides *contexts* to the tags, is important as well. For example, the tag "Apple" given by someone who is interested in digital image processing is likely referring to a computer on which a photo is digitally processed instead of referring to a fruit displayed in the photo.

In this paper we propose CubeLSI, a tag-based resource retrieval technique that addresses the two problems, namely, *tag-ambiguity* and *a multitude of aspects*, as mentioned above. In a nutshell, CubeLSI applies Latent Semantic Indexing (LSI) [9, 10] to capture high-level semantic information of tags to improve the precision of resource retrieval. However, unlike traditional LSI, CubeLSI takes into account the tag-tagger relationship (i.e., which tags are assigned by whom) in performing semantic analysis. As we will see, our approach of taking taggers as a feature dimension significantly improves the quality of the semantic analysis. This results in more accurate resource retrieval. In the following, we further discuss how CubeLSI approaches the two problems. We will also give the general framework of CubeLSI.

Tags, Concepts and Aspects As we have discussed, there are generally a multitude of *aspects* for a given resource. When a tagger tags a resource, he first studies the resource to identify certain aspects of his interest. For example, a tagger may be interested in the type of event (e.g., wedding) behind a picture of a bouquet instead of the kind of flower (e.g., roses) shown. The tagger then discovers the *semantic concepts* exhibited by the resource with respect to each chosen aspect, and expresses these concepts via a set of words (*tags*). Each concept thus corresponds to a semantically coherent group of tags. In our bouquet photo example, we have "type-of-event" being an aspect, "wedding/engagement/marriage" being a concept, and the word, say, "wedding" being a tag to express the concept. Similarly, when issuing a query, a user first comes up with the semantic concepts he is interested in, and then formulates his query by providing a few tags (e.g., marriage). To avoid the tag-ambiguity problem, the matching between queries and resources should be carried out at the semantic concept level. This makes it possible to match a given query with its relevant resources even if the query and the resources are described by disjoint sets of tags. Therefore, our CubeLSI approach transforms the *bag of tags* associated with a resource to a *bag of concepts* to enhance search quality. We call the process of extracting concepts from resources *Concept Distillation.*

Resources, Taggers and Tags LSI is a popular semantic analysis technique that attempts to overcome the word ambiguity problem by extracting concepts from words in an unstructured text collection. Since LSI has proven to be remarkably successful in IR, our CubeLSI approach

[4] `http://blog.last.fm/2009/03/24/lastfm-radio-announcement`

Table 1: Tag pairs and their semantic relations

Tag Pairs	Human-judged	CubeLSI	LSI
⟨comedy, humour⟩ ⟨virus, antivirus⟩ ⟨wireless, WiFi⟩	**Y**	**Y**	**N**
⟨cancer, charities⟩ ⟨shopping, photography⟩ ⟨festival, music⟩	**N**	**N**	**Y**

adapts LSI to extract concepts from tag-based data. In traditional document retrieval systems, LSI performs Singular Value Decomposition (SVD) on a *term-document matrix.* (For a social-tagging system, this matrix corresponds to the tag-resource relation, i.e., which tags are assigned to which resource.) To extract concepts, LSI identifies terms (tags) that are highly semantically related. Essentially, two terms (tags) are related or similar if and only if they have very similar *contexts*, e.g., they occur in similar sets of documents (resources). As we have argued, besides the set of resources, taggers also represent a very important dimension of the tags' feature space in social tagging systems. Therefore, CubeLSI extends LSI by considering the tagger dimension. More specifically, instead of applying SVD on a 2D tag-resource matrix, CubeLSI performs *Tucker Decomposition* on a third-order tensor whose dimensions are resources, taggers, and tags in order to discover semantically related tags.

To further substantiate our claim that the tagger dimension improves tag semantic analysis, we give a preview of some of the empirical observations we made in our experiments. We conducted user studies on the *Delicious* dataset (see Section 6) to compare traditional LSI against CubeLSI in identifying the semantic relations of tags. Our general observation is that the semantic relations obtained by CubeLSI better resemble those identified by human observers. To illustrate, Table 1 shows a few tag pairs together with the results of their semantic relations derived from different schemes. A '**Y**' denotes that two tags are highly semantically related, while an '**N**' denotes that two tags are weakly semantically related. We see that the results under CubeLSI are consistent with those given by humans, whereas the results under LSI are not. These examples show the importance of the tagger dimension (which is used in CubeLSI) in measuring tag similarity.

The CubeLSI Framework Figure 1 shows the CubeLSI framework for tag-based searching of resources in social tagging systems. The framework consists of an offline concept-extraction component and an online query-processing component. For the offline component, tag assignments are represented by a third-order tensor that captures the relationships among resources, taggers, and tags. Tucker decomposition is then applied to the tensor to generate pairwise tag distances (and thus tag similarities) accurately. Concepts are distilled by clustering tags based on their similarities[5]. After that, the tags of each resource are mapped to their corresponding concepts. For the online component, the tags of a given query are similarly transformed into a bag of concepts. These query concepts are then matched against those of the resources using cosine similarity measure. Finally, a list of resources ranked according to the matching results is presented to the user.

Here, we summarize the major contributions of our work:

1. We introduce a novel tag-based domain-free framework for searching resources in social tagging systems. Our framework exploits the special properties of social tagging systems in which taggers are generally untrained causal users and that tags come from an uncontrolled vocabulary. These properties make traditional IR methods less effective than our approach.

2. We study the role of taggers in search quality for social tagging systems. To this end, we

[5] In this paper, we perform hard clustering to assign each tag to one single concept. To address the polysemy problem, a soft-clustering method could be employed, so that each tag may be assigned to multiple concepts with different weights. We are exploring in this direction in our research.

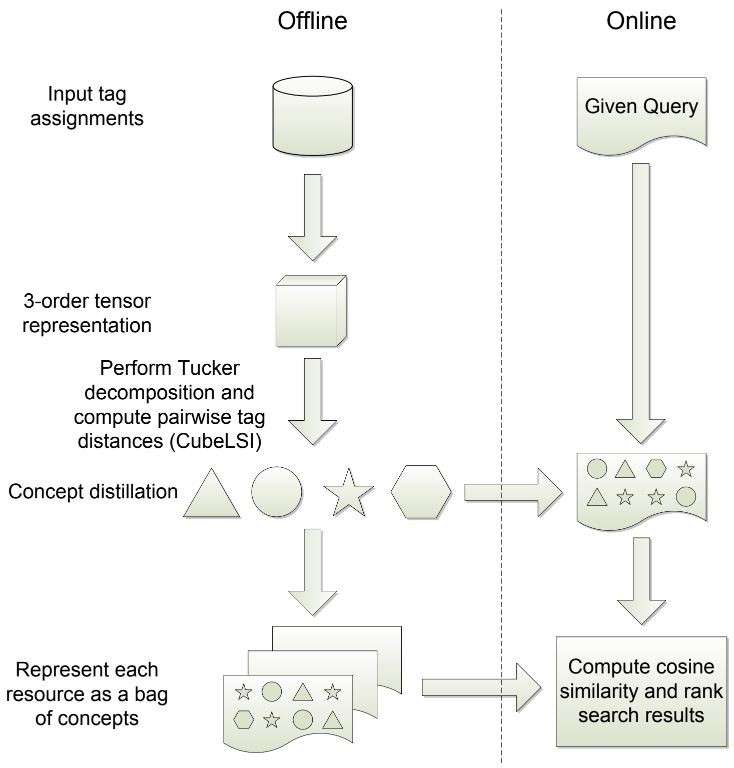

Offline
Online
Input tag assignments
Given Query
3-order tensor representation
Perform Tucker decomposition and compute pairwise tag distances (CubeLSI)
Concept distillation
Represent each resource as a bag of concepts
Compute cosine similarity and rank search results

Figure 1: The CubeLSI framework

employ the notion of *tensor* by which taggers, in addition to tags and resources, are fully captured for analysis.

3. We propose CubeLSI, which is a third-order extension of LSI, for semantic analysis over the third-order tensor of resources, taggers, and tags. We apply Tucker decomposition to the tensor to obtain a purified tensor with which pairwise tag similarities are effectively computed. With the additional tagger dimension and the large volumes of data in social tagging systems, the computation of tag similarities is very expensive. We prove two mathematical theorems that lead to interesting shortcuts in tag similarity computation. This results in an extremely efficient execution of CubeLSI.

4. We present a comprehensive empirical evaluation of CubeLSI against a number of ranking methods on real datasets. The results show that CubeLSI is highly effective and efficient in searching resources in existing social tagging systems.

The rest of the paper is organized as follows. Section 2 summarizes the related work. Section 3 gives the details of how resources are matched and ranked against a user query. Section 4 presents the process of computing pairwise tag distances using three-dimensional semantic analysis. Section 5 discusses how spectral clustering is applied to tags for concept distillation. Section 6 gives the experimental results. Finally, Section 7 concludes this paper.

2 Related Work

Matrix factorization (*MF*) is a research field that is related to our work. MF has been proven to be most successful in building modern collaborative-filtering-based recommender systems [17, 26, 29]. A basic recommender system works on a user-item matrix $W \in \mathbb{R}^{n \times m}$, e.g., a rating matrix in which the entry $W_{i,j}$ represents the rating of item j given by user i. The basic idea of MF is to fit the matrix W with a low-ranked approximation $\hat{W}$ by factorizing it as the product of two matrices such that $W \approx \hat{W} = UM$, where $U \in \mathbb{R}^{n \times k}$ and $M \in \mathbb{R}^{k \times m}$. To find a good approximation $\hat{W}$, we need to minimize a loss measure such as the sum of the squared differences between the known entries in W and their predictions in $\hat{W}$. With the approximation $\hat{W}$, a recommender system can predict the ratings of the unknown entries in the user-item matrix. One possible way to find such an approximation is to perform SVD on the matrix W.

Our work also involves factorization but is significantly different from MF in two ways: (1) We aim at capturing semantic relations among tags rather than predicting unknown entries in the user-item matrix. (2) We deal with a three-dimensional tensor instead of a two-dimensional matrix. A multidimensional generalization of matrix SVD, called Tucker decomposition, was introduced in [19]. The analogy between corresponding properties of SVD and Tucker decomposition, e.g., uniqueness, have been investigated in depth. An efficient algorithm for Tucker decomposition, called MET, was proposed in [16]. MET works well with sparse high-dimensional data with the assumption that there is enough memory to fit the output tensor, which serves to compute pairwise tag distances, resulting from the decomposition. As we will see later, for most social tagging systems, the output tensors are prohibitively large and dense. The assumption MET bases on does not hold in those systems and hence MET cannot be directly applied in our CubeLSI approach. One of our major contributions are two theorems that allow us to overcome the computational issues in the tensor decomposition.

An active research topic on social tagging systems is tag recommendation. The objective is to recommend tags for a user to annotate a given resource. A number of studies [28, 8, 11, 31] have proposed different approaches to recommend tags to Flickr (a photo-sharing service) users. There are also studies that address tag recommendations in more general settings, besides photo-sharing systems [14, 22, 7, 1]. Our study differs from these in that we focus on processing tag-based resource-searching queries.

We remark that the literature on searching relevant resources in social tagging systems remains sparse. The state of the art is FolkRank [12], which can be seen as a modified version of PageRank [4]. First, resources, taggers, and tags are represented by an undirected, weighted, tripartite graph. Then, in a way similar to how PageRank propagates authority, FolkRank iteratively computes vertices' weights according to the formula: $\mathbf{w} \leftarrow dA\mathbf{w} + (1-d)\mathbf{p}$, where A is the row stochastic version of the adjacency matrix of the tripartite graph, $\mathbf{p}$ is the preference vector (also called random surfer) that can be used to express user preferences by giving a higher weight to those tag vertices that appear in the query, and $d \in [0,1]$ is a constant that controls the influence of the random surfer. This weight-propagation scheme follows the assumption that votes cast by important taggers with important tags would make the annotated resources important. A resource that receives a higher weight is considered more relevant to a given query. Note that our CubeLSI approach differs from FolkRank significantly. In particular, CubeLSI performs offline semantic analysis, which allows online query processing to be efficiently done by simply matching the concepts of resources to those of user queries.

Another piece of work that is related to ours is introduced in [30]. This work applies a separable mixture model to demonstrate emergent semantics in a social tagging system. Particularly, in their model, concepts are captured by a latent variable z, which independently generates occurrences of users, tags and resources for a particular triple (u, t, r). The joint distribution over users, tags and resources is defined as $p(u, t, r) = \sum_z p(z)p(u|z)p(t|z)p(r|z)$. The parameters $p(u|z)$, $p(t|z)$, $p(r|z)$ and $p(z)$ are then estimated by maximizing the log-likelihood of social tagging data. While their motivation is similar to ours, they did not quantitatively evaluate their approach in terms of search quality and performance. Apart from that, our approach is technically different from that work.

In a preliminary study, we have briefly assessed the effectiveness of extending LSI to 3D so as to incorporate the dimension of users [3]. The preliminary results are encouraging. However, that prototype system did not scale well to handle large social tagging databases. We have since overcome this hurdle by introducing new performance enhancements, thanks to Theorems 1 and 2, to reduce the execution time of CubeLSI significantly. We are also presenting extensive performance evaluations in Section 6 to show the merits of our approach.

3 Ranking resources against Tag-based Queries

In this section we present the IR model we use for ranking resources given a user query. There have been a number of models for representing documents in information retrieval [21]. Thanks to its efficient implementation with an inverted index, the bag-of-words model has become a standard in text retrieval [2, 25]. In the bag-of-words (tags) model, a document (resource) d is represented as a sparse vector $\mathbf{d}$ of length n, where n is the number of distinct words (tags) in the corpus. The entry $\mathbf{d}[i]$ gives the number of occurrences of word i in document d. As we have discussed, user queries and resources should be matched at the concept level instead of at the tag level to avoid the tag-ambiguity problem. Therefore, we first transform the bag of tags of a resource into a bag of concepts. Likewise, the tags given by a query are similarly transformed. Thus, in our *bag-of-concept* model, each resource or query is represented by a sparse vector of concepts. In the rest of this section, we assume that this tag-to-concept transformation is done. The details of this transformation will be explained clearly in Sections 4 and 5.

We use the *vector space model*, first proposed in [27], to measure the similarity between a resource r and a query q. Let L be the set of distilled concepts. We assign a *tf-idf* weight for each concept l_i for r:

$$w(l_i, r) = \mathit{tf}(l_i, r) \times \log(N/n_{l_i}), \tag{1}$$

where N denotes the total number of resources in the corpus, n_{l_i} denotes the number of resources in which concept l_i occurs, and $\mathit{tf}(l_i, r)$ represents the normalized count of occurrences of concept l_i in resource r. More specifically, $\mathit{tf}(l_i, r)$ is given by

$$\mathit{tf}(l_i, r) = \frac{c(l_i, r)}{\sum_{l_k \in r} c(l_k, r)}, \tag{2}$$

where $c(l_i, r)$ is the occurrence count of concept l_i in resource r. Resource r is then associated with the vector:

$$\mathbf{r} = (w(l_1, r), w(l_2, r), \ldots, w(l_{|L|}, r)). \tag{3}$$

Similarly, a query with one or more tags can also be given a weighted vector over the set of concepts.

With the bag-of-concept model, resources are ranked based on their *cosine similarity* with the query:

$$\cos(q, r) = \frac{\sum_{l \in L} w(l, q) \times w(l, r)}{\sqrt{\sum_{l \in L} w(l, q)^2} \times \sqrt{\sum_{l \in L} w(l, r)^2}}. \tag{4}$$

A list of relevant resources sorted in descending order of their cosine similarity scores is returned.

4 Tag Proximity Assessment

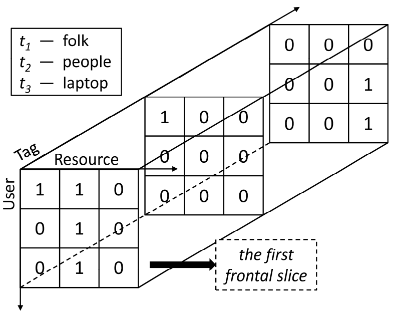
For the purpose of concept distillation, we have to compute pairwise semantic distances between tags first, so as to group semantically similar tags together to form a concept. To compute the tag distances, we propose a new method called CubeLSI. It is a three-dimensional extension of LSI that incorporates the dimensions of users, tags and resources simultaneously into tag semantic analysis.

4.1 Third-order Tensor Representation

There are four types of entities in a social tagging system: a set of users (or taggers) U, a set of tags T, a set of resources R, and a set of tag assignments $Y \subseteq U \times T \times R$. A triple $(u, t, r) \in Y$ denotes that user $u \in U$ has annotated resource $r \in R$ with tag $t \in T$. Figure 2(a) shows a few sample records taken from *Delicious*, a social bookmarking system, on three chosen tags: "*folk*", "*people*", and "*laptop*". Each row represents a record that a user (u) annotated a resource (r) with a tag (t), and thus each record can be represented by a triple $(u, t, r) \in Y$. In this example, the number of users $|U| = 3$, the number of tags $|T| = 3$, the number of resources $|R| = 3$ and the number of tag assignments $|Y| = 7$. User u_1 annotated resource r_1 with tag t_1 (*folk*) and

Record	User	Tag	Resource
1	u_1	t_1	r_1
2	u_1	t_1	r_2
3	u_2	t_1	r_2
4	u_3	t_1	r_2
5	u_1	t_2	r_1
6	u_2	t_3	r_3
7	u_3	t_3	r_3

(a) Sample records taken from a social booking system



(b) Corresponding tensor $\mathcal{F} \in \{0,1\}^{3\times3\times3}$

Figure 2: Data model

Record	Tag	Resource	Value
1	t_1	r_1	1
2	t_1	r_2	3
3	t_2	r_1	1
4	t_3	r_3	2

(a) Records after aggregating the data over the user dimension

Tag \ Resource			
	1	3	0
	1	0	0
	0	0	2

(b) Corresponding matrix $F \in \mathbb{R}^{3\times3}$

Figure 3: Two-dimensional data without the user dimension

tag t_2 (*people*). Tag t_1 was used by all the three users to annotate resource r_2. Users u_2 and u_3 both annotated resource r_3 with tag t_3 (*laptop*).

Different from traditional IR systems in which a collection is represented as a term-document matrix, our CubeLSI technique employs a third-order tensor $\mathcal{F} \in \{0,1\}^{|U|\times|T|\times|R|}$ to represent the tag assignments in a social tagging system.

A *tensor* is a multidimensional array which extends the notion of scalar, vector, and matrix. The value of the entry (u,t,r) of our third-order tensor $\mathcal{F}$ is determined by

$$\mathcal{F}_{u,t,r} = \begin{cases} 1 & \text{if } (u,t,r) \in Y, \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Figure 2(b) visualizes the third-order tensor $\mathcal{F} \in \{0,1\}^{3\times3\times3}$ of the data in Figure 2(a). The entries of $\mathcal{F}$ are determined by equation (5). For instance, $\mathcal{F}_{3,1,2} = 1$ results from the fourth record that user u_3 used tag t_1 to annotate resource r_2. $\mathcal{F}_{:,1,:} \in \{0,1\}^{3\times3}$ denotes the first frontal slice of $\mathcal{F}$. In particular,

$$\mathcal{F}_{:,1,:} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix},$$

which records who has used tag t_1 to annotate which resource. The user-resource matrix $\mathcal{F}_{:,1,:}$ carries exhaustive available information on tag t_1 extracted from the original data. Thus, it can serve as the feature representation of tag t_1. Analogously, the feature representations of tag t_2 and tag t_3 are:

$$\mathcal{F}_{:,2,:} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \text{and } \mathcal{F}_{:,3,:} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}.$$

As a comparison, Figure 3 illustrates the data model used by traditional IR, where users are not discerned. Figure 3(a) lists the records obtained by aggregating the data in Figure 2(a) over the user dimension. For instance, the second record indicates that tag t_1 was assigned to resource r_2 by three users. As shown in Figure 3(b), traditional IR systems model the data in

Table 3(a) by a two-dimensional matrix, denoted as $F \in \mathbb{R}^{|T|\times|R|}$. $F_{t_i,:} \in \mathbb{R}^{|R|}$ is regarded as the feature representation of tag t_i in traditional IR. In particular, $F_{t_1,:} = (1\ 3\ 0)$, $F_{t_2,:} = (1\ 0\ 0)$, and $F_{t_3,:} = (0\ 0\ 2)$. With the vector representation of tags, the pairwise distance between tag t_i and tag t_j is given by computing the L_2 distance between the two vectors:

$$d_{i,j} = ||F_{t_i,:} - F_{t_j,:}||_2. \tag{6}$$

For example, the distance between tag t_1 (*folk*) and tag t_2 (*people*) in vector representation is:

$$d_{1,2} = ||F_{t_1,:} - F_{t_2,:}||_2 = \sqrt{9}. \tag{7}$$

In our approach, full information in the user dimension is considered. Therefore, tags are represented by matrices as mentioned before. This gives greater discerning power to our method, which can help defining semantic distance between tags more accurately. The pairwise distance between tag t_i and tag t_j is given by computing the Frobenius-norm (i.e., matrix norm of a matrix defined as the square root of the sum of the absolute squares of its entries) of the difference between the two matrices:

$$D_{i,j} = ||\mathcal{F}_{:,t_i,:} - \mathcal{F}_{:,t_j,:}||_F. \tag{8}$$

For example, the distance between tag t_2 (*people*) and tag t_3 (*laptop*) in matrix representation is given as:

$$D_{2,3} = ||\mathcal{F}_{:,t_2,:} - \mathcal{F}_{:,t_3,:}||_F = \sqrt{3}. \tag{9}$$

4.2 Why Tensor Decomposition

Before discussing why we extend LSI to a three-dimensional decomposition so as to conduct tag semantic analysis on our tensor, we use the previous example to illustrate the drawbacks in calculating pairwise tag distances without performing the decomposition.

Applying formula 6, we obtain the pairwise distances among tags t_1 (*folk*), t_2 (*people*) and t_3 (*laptop*) in vector representation. Together with (7), we obtain the following inequalities:

$$d_{1,3} = ||F_{t_1,:} - F_{t_3,:}||_2 = \sqrt{14} > \sqrt{9} = d_{1,2} \tag{10}$$

$$d_{2,3} = ||F_{t_2,:} - F_{t_3,:}||_2 = \sqrt{5} < \sqrt{9} = d_{1,2} \tag{11}$$

Notice that, as shown in inequality (11), the distance between tag t_1 (*folk*) and tag t_2 (*people*) is greater than that between tag t_2 and tag t_3 (*laptop*), which is highly counter-intuitive.

Applying formula 8, we compute the pairwise distances among the three tags via matrix representation and obtain the following inequalities:

$$D_{1,3} = ||\mathcal{F}_{:,t_1,:} - \mathcal{F}_{:,t_3,:}||_F = \sqrt{6} > \sqrt{3} = D_{1,2} \tag{12}$$

$$D_{2,3} = ||\mathcal{F}_{:,t_2,:} - \mathcal{F}_{:,t_3,:}||_F = \sqrt{3} = \sqrt{3} = D_{1,2} \tag{13}$$

This time, with $D_{2,3} = D_{1,2}$ in (13), we consider tag t_2 (*people*) to be as different from t_3 (*laptop*) as it is from t_1 (*folk*). This is a slight improvement over the traditional IR approach. At least, we do not conclude that t_2 is closer to t_3 than to t_1. This illustrates that by taking the user dimension into account, we can get a more reasonable inter-tag distance. However, this is still not good enough. It would be more desirable if we can improve the above result further, so that t_2 is closer to t_1 than it is to t_3. In the following, we will develop an improved distance measure $\hat{D}$, which will give $\hat{D}_{2,3} > \hat{D}_{1,2}$.

We have seen that directly including the user dimension into our considering is not sufficient. We attribute such undesirable behavior of $D_{i,j}$ to inevitable noise in the data tensor $\mathcal{F}$. There are two major sources of noise. The first one is that the values of the vast majority of entries of $\mathcal{F}$ are zero. However, most of these zero entries do not really represent a value of zero. In other words, $\mathcal{F}_{u,t,r} = 0$ may not necessarily result from the user u considering the tag t to be completely irrelevant to the resource r. Rather, this zero entry only reflects the fact that user u did not mark r with tag t. This may be because she has not seen r, or that t is not in her vocabulary. Therefore, most zero entries should be interpreted as *missing values* rather than irrelevance. Unfortunately, there is no simple way for us to discern between a missing value and an irrelevance relation between tags, users and resources. Using zero entries to represent missing values is a practical compromise, but this introduces some noise at the same time. The other source of noise results from the fact that tagging is a casual and ad-hoc activity. Taggers may make errors and there is little incentive to ensure an extremely high quality of the tags.

To address this problem, we perform spectral analysis on the tensor $\mathcal{F}$ to extract useful information and to reduce noise. A similar technique has been proven tremendously successful in recommender systems based on matrix factorization [17]. We perform Tensor decomposition [19], the extension of singular value decomposition (SVD) for tensors, to extract the most significant axes of the tensor $\mathcal{F}$, thereby eliminating the noise. As we will see later, the tag semantic analysis leads to appropriate pairwise tag distances, since it discovers the latent factors that govern the semantic correlations among users, tags and resources.

Throughout this paper, we denote tensors by calligraphic letters ($\mathcal{A}$, $\mathcal{B}$, ...), matrices by uppercase letters (A, B, ...), vectors by bold lowercase letters ($\mathbf{a}$, $\mathbf{b}$, ...), and scalars by italic lowercase letters (a, b, ...). The notation $\mathbf{a}_i$ denotes the i-th entry of a vector $\mathbf{a}$. $A_{i,j}$ denotes the entry (i, j) of a matrix A. $\mathcal{A}_{i,j,k}$ denotes the entry (i, j, k) of a third-order tensor $\mathcal{A}$. Let I_1, I_2, and I_3 be the number of users, tags, and resources, respectively, i.e., $I_1 = |U|$, $I_2 = |T|$, and $I_3 = |R|$.

4.3 Tucker Decomposition

Before describing Tucker decomposition, we first familiarize ourselves with the *n-mode product* of a tensor by a matrix.

Definition 1. *The n-mode product of an order-m tensor $\mathcal{F} \in \mathbb{R}^{I_1\times\cdots\times I_m}$ by a matrix $W \in \mathbb{R}^{J_n\times I_n}$, denoted by*

$$\mathcal{G} = \mathcal{F} \times_n W,$$

is another tensor of the same order as $\mathcal{F}$. In particular, $\mathcal{G} \in \mathbb{R}^{I_1\times\cdots\times J_n\times\cdots\times I_m}$, i.e., $\mathcal{G}$ has the same size and shape as $\mathcal{F}$ except in the n-th dimension, whose size is J_n instead of I_n. The values for the entries of $\mathcal{G}$ are given by:

$$\mathcal{G}_{i_1,\ldots,j_n,\ldots,i_m} \triangleq \sum_{i_n=1}^{I_n} \mathcal{F}_{i_1,\ldots,i_n,\ldots,i_m} W_{j_n,i_n}.$$

One may conceptually consider W to be a linear transformation, and the tensor product $\mathcal{F} \times_n W$ to be an application of this transformation onto the n-th dimension of $\mathcal{F}$, turning $\mathcal{F}$ into $\mathcal{G}$. This is akin to multiplying W on the left side of a matrix (i.e., an order-2 tensor) or a vector (i.e., an order-1 tensor). Please refer to [19, 20] for details.

The Tucker decomposition is a factorization of a given tensor $\mathcal{F} \in \mathbb{R}^{I_1\times\cdots\times I_m}$ into the form:

$$\mathcal{F} = \overset{\star}{\mathcal{S}} \times_1 \overset{\star}{Y}^{(1)} \times_2 \overset{\star}{Y}^{(2)} \cdots \times_m \overset{\star}{Y}^{(m)}$$

with $\overset{\star}{\mathcal{S}}$ having the same dimensions as $\mathcal{F}$, and each $\overset{\star}{Y}^{(n)}$ being a unitary $I_n \times I_n$ matrix ($n = 1, \ldots, m$) such that $\overset{\star}{\mathcal{S}}$ satisfies certain conditions (see [19] for details). The tensor $\overset{\star}{\mathcal{S}}$ is called the *core tensor*. Its entries encode the importance of various axes of a transformed space, analogous to the singular values of SVD. The transformed space may be considered to reveal latent semantics of the original data. Each factor matrix $\overset{\star}{Y}^{(n)}$ represents the change of basis in the n-th dimension between the original basis of $\mathcal{F}$ and the transformed space. While SVD lets us extract the principal axes of the linear transformation represented by a matrix, Tucker decomposition lets us extract important information of the multi-linear transformation represented by a tensor.

Note that in the formulation above, the *core tensor* $\overset{\star}{\mathcal{S}}$ has the same dimensions as the original tensor $\mathcal{F}$. In many applications, we would like to reduce memory consumption, as well as to speed up processing by reducing the dimensions of the core tensor. In SVD, a low-rank matrix approximation is obtained by eliminating the row/column vectors that correspond to singular values of small magnitudes. An analogous technique can be used for Tucker decomposition to trim the core tensor $\overset{\star}{\mathcal{S}}$: Consider dimension n, for which we want to shrink $\overset{\star}{\mathcal{S}}$ to size J_n. We identify a subset of indices $P = \{i \mid 1 \le i \le I_n\}$ to be pruned away, such that $|P| = I_n - J_n$. These are the indices that roughly correspond to the tensor entries of small magnitudes. Then, we trim away from $\overset{\star}{\mathcal{S}}$ those entries having the indices in P to get $\overset{\star}{\mathcal{S}}_{(n)} \in \mathbb{R}^{I_1 \times \cdots \times J_n \times \cdots \times I_m}$. The factor matrix $\overset{\star}{Y}^{(n)}$ has to be trimmed accordingly to obtain the trimmed matrix $Y^{(n)}$ of dimension $I_n \times J_n$. (The indices for dimension n of $\overset{\star}{\mathcal{S}}$ (and also the columns of $\overset{\star}{Y}^{(n)}$) are renumbered so that they range from 1 to J_n.) We then get:

$$\hat{\mathcal{F}}_{(n)} = \overset{\star}{\mathcal{S}}_{(n)} \times_1 \overset{\star}{Y}^{(1)} \cdots \times_n Y^{(n)} \cdots \times_m \overset{\star}{Y}^{(m)},$$

which is a tensor with the same dimensions as $\mathcal{F}$. Repeating this for every dimension, we eventually get a fully trimmed core tensor $\mathcal{S} = \overset{\star}{\mathcal{S}}_{(1),\ldots,(m)} \in \mathbb{R}^{J_1 \times \cdots \times J_m}$ and a set of trimmed factors matrices $Y^{(n)} \in \mathbb{R}^{I_n \times J_n}$. Their product

$$\hat{\mathcal{F}} = \mathcal{S} \times_1 Y^{(1)} \cdots \times_n Y^{(n)} \cdots \times_m Y^{(m)} \tag{14}$$

has the same dimensions as $\mathcal{F}$, and is indeed a good approximation of $\mathcal{F}$, retaining the most essential and relevant information from $\mathcal{F}$ [19].

To assess the quality of the approximation, we consider a tensor norm $\|\cdot\|_T$ of the difference introduced by this approximation, i.e. $\|\mathcal{F} - \hat{\mathcal{F}}\|_T$. We use the Frobenius norm for $\|\cdot\|_T$, which is given by:

$$\|\mathcal{F}\|_T = \sqrt{\sum_{i_1=1}^{I_1} \cdots \sum_{i_m=1}^{I_m} (\mathcal{F}_{i_1,\ldots,i_m})^2}. \tag{15}$$

Definition 2 (Tucker decomposition problem). *Given a tensor $\mathcal{F} \in \mathbb{R}^{I_1 \times \cdots \times I_m}$ and reduction ratios $c_n \geq 1$ (for $n = 1, \ldots, m$), we want to find a core tensor $\mathcal{S} \in \mathbb{R}^{J_1 \times \cdots \times J_m}$ and factor matrices $Y^{(n)} \in \mathbb{R}^{I_n \times J_n}$ to:*

$$\begin{aligned} \textit{minimize: } & \|\mathcal{F} - \hat{\mathcal{F}}\|_T \\ \textit{subject to: } & \hat{\mathcal{F}} \triangleq \mathcal{S} \times_1 Y^{(1)} \cdots \times_m Y^{(m)} \\ & c_n = I_n / J_n \ (n = 1, \ldots, m) \end{aligned}$$

We have formulated this problem without mentioning the untrimmed core tensor $\overset{\star}{\mathcal{S}}$. This is because there exist algorithms for solving this problem directly, without first finding $\overset{\star}{\mathcal{S}}$ and then trimming it. In this work, we compute factor matrices $Y^{(n)}$ based on the *alternating least squares* (ALS) principle [20], and then compute the core tensor $\mathcal{S}$ from the factor matrices as follows

$$\mathcal{S} = \mathcal{F} \times_1 (Y^{(1)})^T \times_2 (Y^{(2)})^T \times_3 (Y^{(3)})^T. \tag{16}$$

Note that the reduction ratios c_n play an important role here. With lower reduction ratios, a better approximation (i.e. smaller $\|\mathcal{F} - \hat{\mathcal{F}}\|_T$) can be obtained, at the expense of a bigger core tensor $\mathcal{S}$, which imposes higher demands on CPU cycles and storage space. Higher reduction ratios give core tensors $\mathcal{S}$ of smaller dimensions, thus reduce the amount of computations and memory needed. One may also consider the decomposition as a form of lossy compression of the original tensor $\mathcal{F}$, with the compression ratio controlled by the ratios c_n. The compressed data consist of $\mathcal{S}$ and all $Y^{(n)}$. Decompression is simply the computation of $\hat{\mathcal{F}}$ using equation (14). The higher the compression ratio, the less faithful the reconstructed data $\hat{\mathcal{F}}$. For instance, in the *Last.fm* dataset we use in the experiments (Section 6), we have $I_1 = 3897$, $I_2 = 3326$, $I_3 = 2849$ and hence the tensor $\mathcal{F}$ has 36.9 billion entries. With reduction ratios of $c_1 = c_2 = c_3 = 1$ (i.e. no trimming), we would have 36.9 billion entries in the core tensor $\mathcal{S}$. This would take a lot of time for the decomposition algorithm to run. If we use reduction ratios of $c_1 = c_2 = c_3 = 50$, we would only need to deal with a core tensor $\mathcal{S}$ of dimensions $78 \times 67 \times 57$, which has fewer than 300 thousand entries. This tremendously reduces the computation time and memory required. Yet, the approximation introduced is still quite accurate w.r.t. tag semantics distance derivation. Empirical evaluations demonstrate that our approach leads to competitive search quality even for large reduction ratios.

Table 2: Records after tag semantic analysis on $\mathcal{F}$.

Record	User	Tag	Resource	Value
1	u_1	t_1	r_1	1.19
2	u_1	t_1	r_2	0.92
3	u_2	t_1	r_2	0.92
4	u_3	t_1	r_2	0.92
5	u_1	t_2	r_1	0.36
6	u_1	t_2	r_2	0.28
7	u_2	t_2	r_2	0.28
8	u_3	t_2	r_2	0.28
9	u_2	t_3	r_3	1.00
10	u_3	t_3	r_3	1.00

4.4 Purified Tag Proximity Measure

In Definition 2, we formulated the Tucker decomposition problem as one which is to find an approximation $\hat{\mathcal{F}}$ of a given input tensor $\mathcal{F}$. Do not be mistaken into thinking that $\hat{\mathcal{F}}$ is inferior to $\mathcal{F}$ in terms of the quality of information. Indeed, as mentioned before, we consider the third-order tensor $\mathcal{F}$ obtained from tagging information to be noisy data, containing noise due to missing tags as well as erroneous tags. Indeed, Tucker decomposition is akin to SVD for matrices, and it is a tool for analyzing the multi-linearly mapping represented by the tensor $\mathcal{F}$. The decomposition essentially finds a suitable vector space basis for each dimension of $\mathcal{F}$, and assigns weights to the core tensor $\overset{\star}{\mathcal{S}}$ reflecting the importance of the components of the mapping. When we trim the core tensor, we are only trimming the less important portions of $\mathcal{S}$. What gets removed mainly represents relatively irrelevant, or noisy, information. The remaining entries in $\mathcal{S}$ reflects the more important components of the tensor $\mathcal{F}$. In a nutshell, the resulting tensor $\hat{\mathcal{F}}$ is conceptually a purified version of $\mathcal{F}$, with much noise eliminated.

Since $\hat{\mathcal{F}}$ is considered a purified form of our tag database, we use this tensor to determine inter-tag distance. This gives improved results than using the original data $\mathcal{F}$. In the example in Section 4.1, we characterized tag t_i by the slice $\mathcal{F}_{:,t_i,:}$ of tensor $\mathcal{F}$. Now, we replace it with the corresponding slice $\hat{\mathcal{F}}_{:,t_i,:}$ from the purified tensor $\hat{\mathcal{F}}$ instead. Using this replacement, the *purified* pairwise distance between tags t_i and t_j is:

$$\hat{D}_{i,j} = ||\hat{\mathcal{F}}_{:,t_i,:} - \hat{\mathcal{F}}_{:,t_j,:}||_F. \tag{17}$$

Let us illustrate how the purified tag proximity measure is computed by continuing from the previous example. We start with the tensor given in Figure 2 and perform Tucker decomposition on it. Table 2 shows the non-zero entries in the resulting purified tensor $\hat{\mathcal{F}}$ using $J_1 = 3$, $J_2 = 3$, and $J_3 = 2$. Observe that the entries of $\hat{\mathcal{F}}$ are quite close to those of $\mathcal{F}$. The purified matrix slices $\hat{\mathcal{F}}_{:,t_1,:}$, $\hat{\mathcal{F}}_{:,t_2,:}$ and $\hat{\mathcal{F}}_{:,t_3,:}$ for tag t_1, tag t_2 and tag t_3 are given as follows:

$$\hat{\mathcal{F}}_{:,t_1,:} = \begin{pmatrix} 1.19 & 0.92 & 0.00 \\ 0.00 & 0.92 & 0.00 \\ 0.00 & 0.92 & 0.00 \end{pmatrix}, \ \hat{\mathcal{F}}_{:,t_2,:} = \begin{pmatrix} 0.36 & 0.28 & 0.00 \\ 0.00 & 0.28 & 0.00 \\ 0.00 & 0.28 & 0.00 \end{pmatrix},$$
$$\text{and } \hat{\mathcal{F}}_{:,t_3,:} = \begin{pmatrix} 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 1.00 \\ 0.00 & 0.00 & 1.00 \end{pmatrix}.$$

Now, we can compute the pairwise distances among tag t_1, tag t_2 and tag t_3 using formula (17), giving:

$$\hat{D}_{1,2} = \sqrt{1.92} < \sqrt{5.94} = \hat{D}_{1,3} \tag{18}$$
$$\hat{D}_{1,2} = \sqrt{1.92} < \sqrt{2.36} = \hat{D}_{2,3} \tag{19}$$

So, finally we have $\hat{D}_{1,2} < \hat{D}_{2,3}$, meaning that tag t_2 (*people*) is closer to t_1 (*folk*) than t_3 (*laptop*). This is consistent with intuition. It is an improvement over inequalities (11) and (13).

Thus, by considering the dimension of users and performing latent semantic analysis via Tucker decomposition, we have derived a much better tag-distance measure.

Note that the tag distance measure relies on the tensor $\hat{\mathcal{F}}$ which is dense and huge. To handle computations involving $\hat{\mathcal{F}}$ would require a large amount of memory and calculations. Further, the size of $\hat{\mathcal{F}}$ easily exceeds main memory even on modern machines. So, disk access will be involved, further slowing down the computation. For instance, in the *Last.fm* dataset, there are 36.9 billion entries in $\mathcal{F}$. Each slice $\hat{\mathcal{F}}_{:,t_j,:}$ alone contains 3897 (users) × 2849 (resources) = 11.1 million entries. So, computing the Frobenius norm for each tag pair requires 11.1 million subtractions, squaring and additions. That would take a lot of time, let alone disk access time. Further, there are a total of 3326 tags, giving 5.5 million pairs. The amount of computations needed would be prohibitively huge!

Using formula (17) is thus impractical. Fortunately, there is a short-cut to evaluating $\hat{D}_{i,j}$: Equation (17) can be reformulated as follows:

$$\hat{D}_{i,j} = \sqrt{(Y^{(2)}_{t_i,:} - Y^{(2)}_{t_j,:})\Sigma(Y^{(2)}_{t_i,:} - Y^{(2)}_{t_j,:})^T}, \tag{20}$$

where Σ is a matrix that can be readily computed from the core tensor $\mathcal{S}$. The proof of the reformulation is given as Theorem 1 in the Appendix. Notice that this new formula depends only on the core tensor $\mathcal{S}$ and the factor matrix $Y^{(2)}$. Knowledge of these is sufficient for computing all $\hat{D}_{i,j}$. For our purpose, there is no need to compute any entries of $\hat{\mathcal{F}}$. So, the computation time and storage required for $\hat{\mathcal{F}}$ are eliminated altogether. We only need to store $\mathcal{S} \in \mathbb{R}^{J_1 \times J_2 \times J_3}$ and $Y^{(2)} \in \mathbb{R}^{I_2, J_2}$. Since $J_n = I_n/c_n$, if we choose large reduction ratios c_n, we can save a lot of space. Also, the relatively low dimensions of $\mathcal{S}$ and $Y^{(2)}$ implies fewer computations needed when evaluating with equation (20). For each tag pair, the $O(J_2 J_2)$ runtime complexity of formula (20) is very attractive compared with $O(I_1 I_3)$ of formula (17) when $J_2 \ll I_1$ and $J_2 \ll I_3$.

Take the *Last.fm* dataset as an example and let $c_1 = c_2 = c_3 = 50$. Then, Σ has dimensions of only 67 × 67 and, $Y^{(2)}$ is a 3326 × 67 matrix, containing around 223,000 entries. These sizes can be easily handled by modern computers. Using (20) to evaluate one $\hat{D}_{i,j}$ involves only $2 \times 67 \times 67 = 8978$ multiplications and additions, which is tremendously fewer than 11.1 million subtractions and squarings needed for (17). Multiply this by the number of tag pairs (5.5 million), and the saving is very significant.

4.5 CubeLSI Algorithm

We have devised the CubeLSI algorithm (Algorithm 1) to implement the ideas mentioned above. The input is a third order tensor $\mathcal{F}$ representing the tag data. The caller also needs to specify the dimensions of the core tensor $\mathcal{S}$ in the Tucker decomposition step. The algorithm computes pairwise semantic distances $\hat{D}_{i,j}$ between tags as output. The first step of the algorithm is to invoke the ALS algorithm [20] (Algorithm 2 in the Appendix) to perform Tucker decomposition. This algorithm returns the core tensor and the factor matrices of the decomposition. In addition, a matrix Λ_2 is returned as a side product. This is used in the next step.

In the second step, semantic distance between tag pairs are computed, based on (20). As an optimization, we have substituted this formula with (21). The new formula replaces the matrix Σ with $((\Lambda_2)_{1:J_2,1:J_2})^2$, taking advantage of the availability of Λ_2, which is a by-product of the ALS algorithm. The proof of the correctness of this substitution is stated as Theorem 2 in the Appendix.

Note that we do not compute the purified tensor $\hat{\mathcal{F}}$ at all, because it is unnecessary and impractical. Indeed, $\hat{\mathcal{F}}$ is a dense matrix and even the scalable tensor decomposition technique proposed in [16] becomes unusable because it is designed for sparse tensors. The huge amount of computations and storage required for $\hat{\mathcal{F}}$ would render the whole method useless.

5 Concept Distillation

With the pairwise semantic distances computed between tags based on the new tag representation, an off-line clustering is performed to distill a set of concepts from the tags. The discovered tag clusters, each of which is regarded as a concept, form the basis of our ranking model. In

Algorithm 1: CubeLSI

```
Input: tensor $\mathcal{F} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, target core tensor dimensions $J_1, J_2, J_3$
Output: semantic distance $\hat{D}_{i,j}$ between tag $t_i$ and tag $t_j$, where $1 \le t_i, t_j \le I_2$
// Tucker decomposition
$(\mathcal{S}, Y^{(1)}, Y^{(2)}, Y^{(3)}, \Lambda_2) \leftarrow \texttt{ALS}(\mathcal{F}, J_1, J_2, J_3)$
// Pairwise tag distance computation
for $t_i \leftarrow 1$ to $I_2$ do
    for $t_j \leftarrow t_i + 1$ to $I_2$ do
        $X \leftarrow Y^{(2)}_{t_i,:} - Y^{(2)}_{t_j,:}$
        $\hat{D}_{i,j} \leftarrow \sqrt{X((\Lambda_2)_{1:J_2,1:J_2})^2 X^T}$        (21)
    end
end
```

this paper we use spectral clustering [23], which takes as input a distance matrix for partitioning the set of tags into groups of semantically related tags. We describe here the key steps of the algorithm:

1. Transform the distance matrix $\hat{D} \in \mathbb{R}^{|T| \times |T|}$ over a set of tags to an affinity matrix $A \in \mathbb{R}^{|T| \times |T|}$ by $A_{i,j} = \exp(-\hat{D}^2_{i,j}/\sigma^2)$ if $i \neq j$, 0 if $i = j$.
2. Construct the diagonal matrix M, with $M_{i,i} = \sum_{j=1}^{|T|} A_{i,j}$, and form the matrix $L = M^{-1/2} A M^{-1/2}$.
3. Create a matrix $X \in \mathbb{R}^{|T| \times k}$ from the vectors associated with the k largest eigenvalues of L. k is chosen either by stipulation or by picking sufficient eigenvectors to cover 95% of the variance. Then, normalize each of X's rows to have unit length.
4. Treating each tag as a row of X with k entries, apply the k-means algorithm to cluster the tags into semantically coherent groups. Each group is considered a concept.

Using the values from (18) and (19) derived from our running example, we carry out the above clustering process with $\sigma = 1$ and $k = 2$. Then, we obtain:

$$A = \begin{pmatrix} 1 & 0.147 & 0.00263 \\ 0.147 & 1 & 0.0944 \\ 0.00263 & 0.0944 & 1 \end{pmatrix} \quad M = \begin{pmatrix} 1.15 & 0 & 0 \\ 0 & 1.24 & 0 \\ 0 & 0 & 1.10 \end{pmatrix}$$

$$L = \begin{pmatrix} 0.870 & 0.123 & 0.00234 \\ 0.123 & 0.806 & 0.0809 \\ 0.00234 & 0.0809 & 0.912 \end{pmatrix} \quad X = \begin{pmatrix} -0.614 & 0.574 \\ -0.140 & 0.597 \\ 0.777 & 0.561 \end{pmatrix}$$

After normalizing the row vectors of X and performing k-means clustering on them, the tags t_1 (*folk*) and t_2 (*people*) are grouped into one cluster, whereas t_3 (*laptop*) by itself forms another cluster. This result makes sense semantically.

In addition to benefiting resource search, tag clustering enables users to explore the tag space in social tagging systems conveniently and effectively. By looking at semantically cohesive tag clusters, users are likely to discover useful information that search engines might not provide. They are probably inspired by exploring the related tags especially when their information needs are not well defined. Social tagging systems could improve their user experience by grouping semantically related tags. As shown in Section 6, our approach is capable of discovering not only synonymous tags, but also latent semantic relatedness between tags.

6 Empirical Evaluation

In this section we present the experimental evaluation of CubeLSI. We first describe the real datasets used in the experiments in Section 6.1. In Section 6.2 we briefly describe five other

Table 3: Dataset statistics

Dataset		$\|U\|$	$\|T\|$	$\|R\|$	$\|Y\|$
Delicious	raw	326,526	171,584	56,452	3,398,402
	cleaned	28,939	7,342	4,118	1,357,238
Bibsonomy	raw	3,655	70,470	305,754	1,083,512
	cleaned	732	4,702	35,708	258,347
Last.fm	raw	31,828	29,105	33,790	1,455,309
	cleaned	3,897	3,326	2,849	335,782

ranking methods against which CubeLSI is compared. The methods are evaluated on three metrics: (1) The accuracy of the pairwise tag distances derived (Section 6.3), (2) the quality of the ranked list of resources returned given a query workload (Section 6.4), and (3) the time/space efficiency of the methods.

6.1 Datasets

We conduct experiments on data collected from three social tagging systems, namely, *Delicious*, *Bibsonomy* and *Last.fm*. *Delicious* is a social bookmarking system which allows users to bookmark their favorite URLs with descriptive tags. In addition to bookmarks, *Bibsonomy* allows users to organize and tag publications. *Last.fm* is a music sharing website where users can annotate artists, albums and tracks with tags.

The three raw datasets are noisy and very sparse. To clean the data, we first remove system-generated tags (e.g., "*system:imported*", "*system:unfiled*", etc.). Then, we follow a similar approach applied in [14] to eliminate outliers[6]. We also convert all tag letters into lowercase. Table 3 shows some statistics of the datasets.

6.2 Other Ranking Methods

Besides CubeLSI, here we briefly describe five other ranking methods that return a ranked list of resources given a query q. We will compare CubeLSI against these five methods later in this section.

Freq Given a query q and a resource r, one way to measure their similarity using the taggers' information is to ask the following question: "If a user tags r, how likely does he use some tags in q to do so?" Intuitively, the higher this likelihood, the more relevant r is to q. Formally, given a tag-assignment (U, T, R, Y), let $q \subseteq T$ be a query, $\mathit{tags}(r) = \{t|(u,t,r) \in Y, u \in U, t \in T\}$ be the tag set of resource r, and $\mathit{users}(t,r) = \{u|(u,t,r) \in Y, u \in U\}$ be the set of users who annotate resource r with tag t. Freq measures the similarity, $\mathit{Sim}_{\mathit{freq}}(q,r)$, of q and r by,

$$\mathit{Sim}_{\mathit{freq}}(q,r) = \begin{cases} 0 & \text{if } \mathit{tags}(r) = \emptyset, \\ \frac{\sum_{t \in q \bigcap \mathit{tags}(r)} |\mathit{users}(t,r)|}{\sum_{t \in \mathit{tags}(r)} |\mathit{users}(t,r)|} & \text{if } \mathit{tags}(r) \neq \emptyset. \end{cases}$$

Note that $\mathit{Sim}_{\mathit{freq}}(q,r)$ ranges from 0 to 1. Resources are ranked in decreasing similarity values.

BOW (Bag-of-Words) This method uses the traditional vector space model in document retrieval systems by regarding each resource as a document and each tag as a word. Similar to the description we presented in Section 3, queries and resources are represented by vectors of *tf-idf* weights, except that the weights are defined on tags instead of on concepts. Hence, semantic analysis (i.e., concept-distillation and tag-to-concept mapping) is not performed under BOW. Cosine similarity is used to rank resources for a given query.

[6] In our experiments, a user, a tag or a resource is deleted if it appears in less than 5 assignments. By going through the deleted tags, we found that most of them were either incomprehensible gibberish or very specific terms that are rarely used. The latter could be handled by maintaining a small structure containing such terms and their tagged resources. We have conducted experiments both with and without the removal of rarely occurring users or resources. We found that there was little impact on our tag semantic analysis.

FolkRank This method uses FolkRank, which was designed particularly for searching resources in social tagging systems, to rank resources. We have already described FolkRank in Section 2. Readers are referred to [12] for more details on FolkRank.

LSI This method projects the third-order tensor $\mathcal{F}$ (see Equation 5) onto a 2D tag-resource matrix. The user (tagger) dimension is thus removed. The method then applies traditional LSI on the tag-resource matrix using SVD to obtain latent concepts. Essentially, LSI is the same as CubeLSI except that the user (tagger) dimension is ignored. We have claimed that the tagger dimension provides important information which can be used to improve resource retrieval result. It is thus interesting to compare CubeLSI, which uses the tagger dimension, against LSI, which does not.

CubeSim In Sections 4.2, 4.3 and 4.4, we argued that the application of Tucker decomposition to obtain a purified tensor ($\hat{\mathcal{F}}$) allows pairwise tag distances to be more accurately captured. It is thus interesting to see how resource retrieval is affected if those steps are skipped. Our next method, CubeSim, is similar to CubeLSI except that it computes the distance between two tags t_i and t_j directly from the tensor $\mathcal{F}$ by $D_{t_i,t_j} = ||\mathcal{F}_{:,t_i,:} - \mathcal{F}_{:,t_j,:}||_F$, where $\mathcal{F}_{:,t_i,:}$ and $\mathcal{F}_{:,t_j,:}$ are the tensor slices that correspond to t_i and t_j, respectively, and $|| \cdot ||_F$ denotes the Frobenius-norm.

6.3 Tag Semantic Relations

Recall that concept distillation is an important aspect of the CubeLSI approach. In order to identify latent concepts, tags are clustered. The quality of tag clusters thus has a significant impact on the concepts extracted. An accurate measure of tag distances, which leads to high-quality tag clustering, is therefore very important. Among the six methods we consider, LSI, CubeSim, and CubeLSI perform semantic analysis and therefore they compute pairwise tag distances. To reiterate, LSI ignores the tagger dimension and performs decomposition on the tag-resource matrix; CubeSim includes the tagger dimension but does not perform decomposition on the third-order tensor $\mathcal{F}$; CubeLSI performs Tucker decomposition on $\mathcal{F}$ and obtains a purified tensor $\hat{\mathcal{F}}$ based on which tag distances are computed. In our first experiment, we compare the above three methods in terms of the *accuracy* of the tag distances they derive.

To evaluate accuracy, we must have ground truth. For that, we follow the evaluation strategy employed in [18] and use WordNet as our ground truth[7]. In this experiment, we use the Bibsonomy dataset. Since not all tags in Bibsonomy appear in WordNet, we focus only on those tags that do appear in WordNet. Let us call this subset of tags D. In our experiment, there are 2,365 tags in D, which represent 50.3% of all tags in the Bibsonomy dataset. Given two tags t_1, t_2 in D, we use the Jiang-Conrath distance measure [15], $JCN(t_1, t_2)$, which combines taxonomic and information-theoretic knowledge, as the reference true distance between t_1 and t_2.

To evaluate a method, we carry out the following procedure. For each tag $t \in D$ we find its *most similar tag* t_{sim} in Bibsonomy, which is the one that gives the smallest tag distance from t as measured by the method. t_{sim} is thus the most semantically related tag of t as judged by the method. If t_{sim} is in WordNet, we compute the JCN distance between t and t_{sim}. We repeat this procedure for every tag $t \in D$ and compute the average JCN distance, given by:

$$JCN_{avg} = \frac{\sum JCN(t, t_{sim})}{k}, \tag{22}$$

where k denotes the number of tags t in D whose t_{sim} is present in WordNet. Intuitively, the smaller JCN_{avg} is, the better is the method in identifying the best semantically related tags. This reflects the accuracy of the pairwise tag distances the method derives.

Another way to evaluate a method is based on an *average rank score*, which is computed as follows. For each tag $t \in D$, again we find its t_{sim} using the method. If t_{sim} is in WordNet, we determine its *rank*, denoted by $Rank(t, t_{sim})$, among all the tags (except t) in WordNet according

[7] WordNet is a semantic lexicon of the English language. There have been a number of methods that use the link structure of WordNet to make semantic distance judgements. Jiang and Conrath introduce the JCN distance in WordNet [15], which combines the taxonomic path length with an information-theoretic similarity measure developed by Resnik [24]. The JCN distance has been empirically validated by user studies, such as [5].

Table 4: JCN_{avg} and $Rank_{avg}$ under different methods

	CubeLSI	CubeSim	LSI
Average JCN	10.32	11.25	11.62
Average Rank	12.55	15.69	16.06

Table 5: Sample tag clusters

Type of Correlation	Tags
synonyms (music-related)	audio, mp3, songs, music
synonyms (source-related)	opensource, open_source, code
synonyms (movie-related)	movie, films, youtube, video
synonyms (England-related)	england, britain, uk
synonyms (photo-related)	photo, photos, foto, flickr
cognates (cross-language)	dictionary, dictionnaire
inflection & derivation	quote, quotes, quotation
abbreviations	ad, advertisement

to their JCN distances from t. For example, if $Rank(t, t_{sim}) = 10$, then the most semantically related tag t_{sim} as identified by the method gives the 10th smallest JCN distance from t among the 2,364 tags in D. Since WordNet with JCN distance is taken as the ground truth, a smaller rank implies a better estimate of the tag distance. In particular, if $Rank(t, t_{sim}) = 1$, then both the method and JCN (ground truth) identify the same tag as the most similar one to t. We compute the rank scores for all $t \in D$ (whose t_{sim} is also in WordNet) and calculate an average:

$$Rank_{avg} = \frac{\sum Rank(t, t_{sim})}{k}. \tag{23}$$

Table 4 shows the average JCN distances and the average rank scores under the three different methods. From the table, we see that CubeLSI gives the smallest average JCN distance, which translates into the lowest (best) average rank score. This implies that CubeLSI is the most accurate in deriving tag distances compared with CubeSim and LSI.

To further study the accuracy of CubeLSI in measuring tag distances, we inspected the tag clusters generated by CubeLSI and found that CubeLSI is effective in identifying semantically related tag clusters. Table 5 shows some illustrative examples of tag clusters acquired by applying CubeLSI on the *Delicious* dataset. We observe that highly related tags are accurately aggregated. For instance, "*audio*", "*mp3*" and "*music*" are correctly grouped together into a music-related tag cluster. Moreover, CubeLSI identifies cognates, e.g., "*dictionary*" in English vs. "*dictionnaire*" in French. In addition to synonyms, CubeLSI is able to discover latent semantic relatedness between tags. "*YouTube*", a video sharing web service, and "*movie*", a form of entertainment, are not synonymous tags, but they are closely related. Also, tags which are morphological variations of the same root word are correctly aggregated, e.g., "*quotation*" vs. "*quotes*". The last row of Table 5 demonstrates its ability to discover abbreviations.

6.4 Ranking Quality

Given a query q, a ranking method returns a ranked list L of resources that are relevant to q. Our next experiment studies the quality of the ranked lists returned by the six ranking methods. We use normalized discounted cumulative gain (NDCG) [13, 6] as the performance metric. NDCG is a measure devised specifically for search result evaluation. Given a ranked list L, NDCG rewards more heavily to relevant resources that are top-ranked in L than those that appear lower down in the list. The *NDCG@N score* is computed as:

$$NDCG@N = Z_N \sum_{i=1}^{N} (2^{r(i)} - 1)/\log(i+1), \tag{24}$$

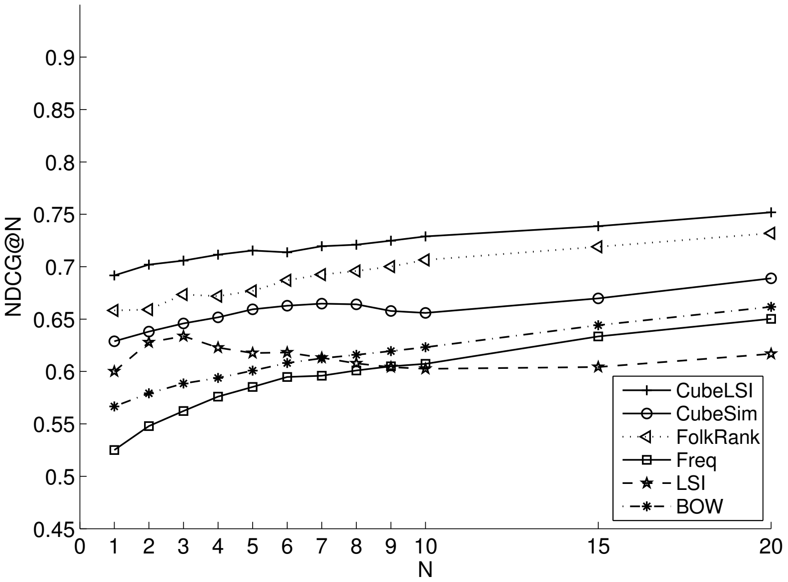


Figure 4: NDCG scores of different ranking methods for the *Delicious* dataset

where @N denotes that the metric is evaluated only on the resources that are ranked top N in list L, $r(i)$ is the *relevance level* (to be discussed shortly) of the resource ranked i in L, and Z_N is a normalization factor that is chosen so that the optimal ranking's NDCG score is 1.

We invited 16 users to participate in this experiment. Each user proposed eight queries. There were thus a total of 128 unique queries. The users then determined a score for each resource returned by the ranking methods by labeling the resource with one of three relevance levels: *Relevant* (score 2), *Partially Relevant* (score 1) and *Irrelevant* (score 0). The overall NDCG score of a ranking method is given by the average of its NDCG scores over the 128 queries.

Figures 4–6 shows the NDCG scores of the six ranking methods at different values of N for the three datasets. For CubeLSI, we set the reduction ratios $c_1 = c_2 = c_3 = 50$. From the figures, we see that CubeLSI, CubeSim, and FolkRank, which use the user (tagger) dimension, consistently outperform the other three methods that consider only tags and resources. This verifies our hypothesis that incorporating tagger information in resource ranking brings about notable improvement in ranking quality. Moreover, CubeLSI gives the best ranking quality among all six ranking methods. This suggests that the three-dimensional tag semantic analysis performed by CubeLSI is very effective in distilling latent concepts from data in social tagging systems.

To illustrate the quality of the ranked lists returned by CubeLSI, Table 9 in the appendix presents a few sample queries of the *Delicious* dataset and the top-5 resources (web sites) returned by CubeLSI for each query. To help readers understand these web sites, we show their URLs and their corresponding ODP (Open Directory Project)[8] categories.

6.5 Efficiency

Having shown that CubeLSI gives the best ranking quality among the various ranking methods, our next set of experiments focuses on the efficiency aspects of CubeLSI. In particular, how it compares against the other two tagger-cognizant methods, CubeSim and FolkRank.

As we have discussed, our CubeLSI approach consists of an offline pre-processing component and an online query-processing component (see Figure 1). We study these two components separately. We note that both CubeLSI and CubeSim have to pre-process the tag-assignment data and to compute pairwise tag distances for concept distillation. We will, therefore, compare CubeLSI and CubeSim in terms of pre-processing time. FolkRank, on the other hand, does not

[8] `http://www.dmoz.org/`, by its own account, is "the largest, most comprehensive human-edited directory of the Web".

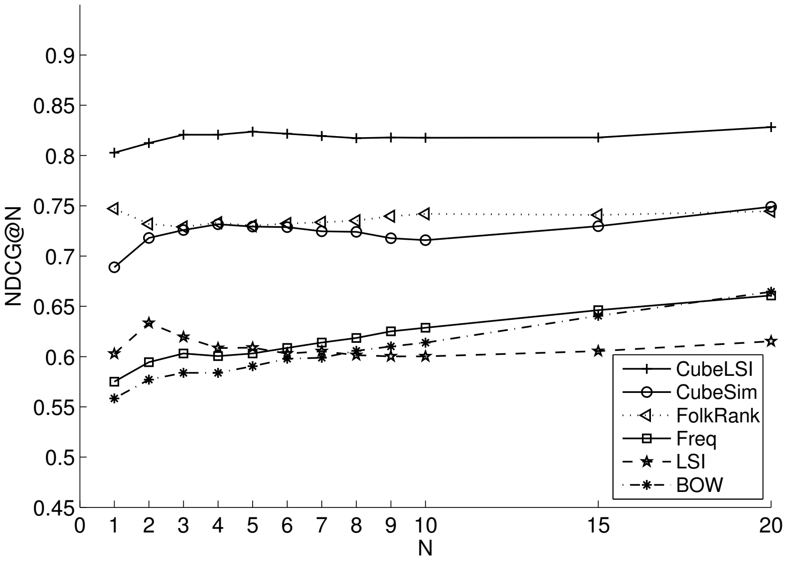


Figure 5: NDCG scores of different ranking methods for the *Bibsonomy* dataset

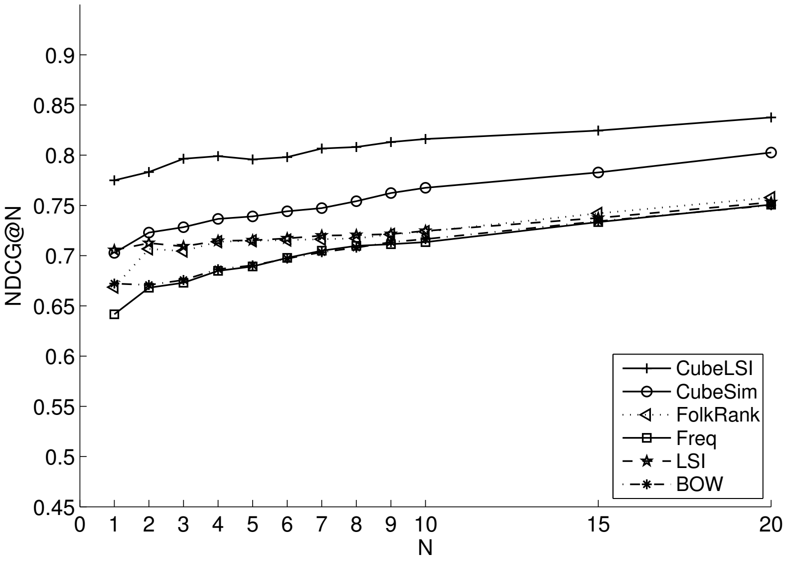


Figure 6: NDCG scores of different ranking methods for the *Last.fm* dataset

Table 6: Pre-processing times (in hours) of CubeLSI and CubeSim

	Delicious	*Bibsonomy*	*Last.fm*
CubeSim	>100	6.29	5.50
CubeLSI	4.55	0.62	0.36

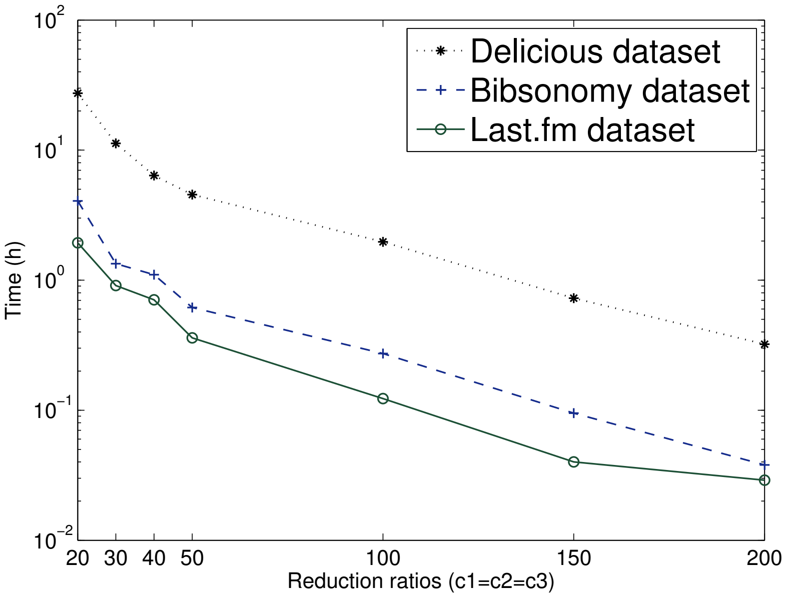


Figure 7: CubeLSI pre-processing time vs. reduction ratios

perform much data pre-processing. Instead, FolkRank performs iterative weight propagation over a tripartite graph. (see Section 2). For CubeLSI, query processing involves computing relatively simple cosine similarity. We will compare CubeLSI and FolkRank in query-processing time. The experiments were conducted on a machine with a 2.6GHz Xeon processor with 8GB memory. We again set the reduction ratios for CubeLSI to $c_1 = c_2 = c_3 = 50$.

Table 6 shows the pre-processing times of CubeLSI and CubeSim over the three datasets. The pre-processing time of CubeSim on the *Delicious* dataset is unavailable because the computation did not finish within 100 hours. From the table, we see that CubeLSI is much faster than CubeSim in data pre-processing. The reason why CubeSim is slow is that it computes the distance between every tag pair, say t_i and t_j, by computing $||\mathcal{F}_{:,t_i,:} - \mathcal{F}_{:,t_j,:}||_F$ from the very big tensor slices (user-resource matrices) $\mathcal{F}_{:,t_i,:}$ and $\mathcal{F}_{:,t_j,:}$. These tag distance computations are therefore very expensive. On the other hand, our theorems allow CubeLSI to compute tag distances from the core tensor $\mathcal{S}$ and the factor matrix $Y^{(2)}$, which are reduced from $\hat{\mathcal{F}}$ given the reduction ratios (c_1, c_2 and c_3). Since S and $Y^{(2)}$ are much smaller structures compared with the tensor slices used by CubeSim, CubeLSI compute tag distances much more efficiently. By varying the reduction ratios, CubeLSI can control the sizes of S and $Y^{(2)}$, which in turn affects the pre-processing time required. Figure 7 shows the pre-processing time of CubeLSI applied on the *Bibsonomy* dataset at different reduction ratios. While setting appropriate values for the reduction ratios is an engineering effort, our experience with many real social tagging systems indicates that reduction ratios of around 50 strike a good balance between efficiency (pre-processing time and storage cost) and ranking result quality.

Table 7 shows the total query-processing times of CubeLSI and FolkRank over the 120 queries. As we can see, CubeLSI is orders-of-magnitude faster than FolkRank in query processing. This is because FolkRank has to perform expensive iterative weight propagation over very large tripartite graphs, which includes vertices of all resources, tags, and users. For CubeLSI, only relatively simple vector dot-products are required in computing cosine similarities.

Finally, we study the memory requirement of CubeLSI. Recall that the basic idea of CubeLSI is to apply Tucker decomposition on the third-order tensor $\mathcal{F}$ to obtain an output tensor $\hat{\mathcal{F}}$,

Table 7: Query-processing times (seconds) of CubeLSI and FolkRank

	Delicious	*Bibsonomy*	*Last.fm*
FolkRank	109.	27.2	25.6
CubeLSI	0.688	1.97	0.323

Table 8: Memory requirements of $\hat{\mathcal{F}}$ vs. $\mathcal{S}$ and $Y^{(2)}$

	Delicious	*Bibsonomy*	*Last.fm*
$\hat{\mathcal{F}}$	7.0 TB	98 GB	88 GB
$\mathcal{S}$ and $Y^{(2)}$	8.8 MB	3.0 MB	1.8 MB

based on which tag distances are computed. We have explained in Section 4.4 that, for a typical social tagging system, the tensor $\hat{\mathcal{F}}$ is so huge that not even the state-of-the-art decomposition algorithm (such as MET [16]) can handle. Again, our two theorems allow CubeLSI to compute tag distances from two relatively small structures $\mathcal{S}$ and $Y^{(2)}$ without ever materializing $\hat{\mathcal{F}}$. The memory requirement of CubeLSI is thus much reduced. Table 8 shows the sizes of $\hat{\mathcal{F}}$ versus the sizes of $\mathcal{S}$ and $Y^{(2)}$ for the three datasets. The table shows that our theorems lead to a very small memory requirement of CubeLSI, which is otherwise infeasible due to the gigantic storage needed.

7 Conclusion

In this paper we proposed and studied the problem of tag-based searching of resources in social tagging systems. We pointed out that, unlike traditional IR systems, social tagging systems involve casual users who label resources with descriptive tags. We observed that the involvement of causal taggers led to two distinctive properties of such systems, namely, *noisy tags* and *multitude of aspects*. To improve ranking quality, we proposed CubeLSI, a ranking algorithm that performed semantic analysis based on user/tag/resource information. CubeLSI was based on Tucker decomposition applied on a third-order tag-assignment tensor. We observed that a straightforward application of the decomposition on the tensor resulted in an output tensor that was too big to be practically useful. We proved two important theorems based on which computational shortcuts were derived to achieve very efficient tag distance computations. We showed that these distances accurately captured the semantic relationships among tags, which led to high-quality concepts. We applied CubeLSI on a number of real datasets and showed that it gave the best ranking quality compared against five other methods. Finally, we showed that CubeLSI was both computationally and storage efficient.

References

[1] Benjamin Adrian, Leo Sauermann, and Thomas Roth-Berghofer. Contag: A semantic tag recommendation system. In *Proceedings of I-Semantics' 07*, pages 297–304, 2007.

[2] Luiz Barroso, Jeffrey Dean, and Urs Hoelzle. Web search for a planet: The Google cluster architecture. *Micro, IEEE*, 23(2):22–28, 2003.

[3] Bin Bi, Lifeng Shang, and Ben Kao. Collaborative resource discovery in social tagging systems. In David Wai-Lok Cheung, Il-Yeol Song, Wesley W. Chu, Xiaohua Hu, and Jimmy J. Lin, editors, *CIKM*, pages 1919–1922. ACM, 2009.

[4] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, April 1998.

[5] Alexander Budanitsky and Graeme Hirst. Evaluating wordnet-based measures of lexical semantic relatedness. *Comput. Linguist.*, 32(1):13–47, 2006.

[6] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *ICML '05: Proceedings of the 22nd International Conference on Machine Learning*, pages 89–96, 2005.

[7] Andrew Byde, Hui Wan, and Steve Cayzer. Personalized tag recommendations via tagging and content-based similarity metrics. In *ICWSM '07*, 2007.

[8] Hong-Ming Chen, Ming-Hsiu Chang, Ping-Chieh Chang, Ming-Chun Tien, Winston H. Hsu, and Ja-Ling Wu. Sheepdog: group and tag recommendation for Flickr photos by automatic search-based learning. In *MM '08: Proceeding of the 16th ACM International Conference on Multimedia*, pages 737–740, 2008.

[9] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41:391–407, 1990.

[10] Susan T. Dumais. Latent semantic indexing (lsi): Trec-3 report. In *Overview of the Third Text REtrieval Conference*, pages 219–230, 1995.

[11] Nikhil Garg and Ingmar Weber. Personalized, interactive tag recommendation for Flickr. In *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems*, pages 67–74, 2008.

[12] Andreas Hotho, Robert Jaschke, Christoph Schmitz, and Gerd Stumme. Information retrieval in folksonomies: Search and ranking. *The Semantic Web: Research and Applications*, 4011:411–426, June 2006.

[13] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4):422–446, 2002.

[14] Robert Jaschke, Leandro Marinho, Andreas Hotho, Lars Schmidt-Thieme, and Gerd Stumme. Tag recommendations in folksonomies. In *Proceedings of the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pages 506–514, 2007.

[15] Jay J. Jiang and David W. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. In *Proceedings of the International Conference on Research in Computational Linguistics*, pages 19–33, 1997.

[16] Tamara G. Kolda and Jimeng Sun. Scalable tensor decompositions for multi-aspect data mining. In *ICDM '08: Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, pages 363–372, 2008.

[17] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[18] Christian Körner, Dominik Benz, Markus Strohmaier, Andreas Hotho, and Gerd Stumme. Stop thinking, start tagging: Tag semantics emerge from collaborative verbosity. In *WWW '10: Proceedings of the 19th International World Wide Web Conference*, pages 521–530. ACM, 2010.

[19] L. D. Lathauwer, B. D. Moor, and J. Vandewalle. A multilinear singular value decomposition. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1253–1278, 2000.

[20] L. De Lathauwer, B. De Moor, and J. Vandewalle. On the best rank-1 and rank-(r1,r2, . . . ,rn) approximation of higher-order tensors. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1324–1342, 2000.

[21] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze. *Introduction to Information Retrieval.* Cambridge University Press, 2008.

[22] Leandro Balby Marinho and Lars Schmidt-Thieme. Collaborative tag recommendations. *Proceedings of 31st Annual Conference of the Gesellschaft für Klassifikation (GfKl) 2007*, 2007.

[23] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14*, pages 849–856, 2001.

[24] Philip Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *IJCAI'95: Proceedings of the 14th international joint conference on Artificial intelligence*, pages 448–453, 1995.

[25] Knut Magne Risvik, Yngve Aasheim, and Mathias Lidal. Multi-tier architecture for web search engines. In *LA-WEB '03: Proceedings of the First Conference on Latin American Web Congress*, page 132, 2003.

[26] Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. In *NIPS '07: Advances in Neural Information Processing Systems 20*, 2007.

[27] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, 1975.

[28] Borkur Sigurbjornsson and Roelof van Zwol. Flickr tag recommendation based on collective knowledge. In *WWW '08*, pages 327–336, 2008.

[29] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Scalable collaborative filtering approaches for large recommender systems. *Journal of Machine Learning Research*, 10:623–656, 2009.

[30] Xian Wu, Lei Zhang, and Yong Yu. Exploring social annotations for the semantic web. In *Proceedings of the 15th international conference on World Wide Web*, pages 417–426, 2006.

[31] Zhichen Xu, Yun Fu, Jianchang Mao, and Difu Su. Towards the semantic web: Collaborative tag suggestions. In *Proceedings of Collaborative Web Tagging Workshop at 15th International World Wide Web Conference*, 2006.

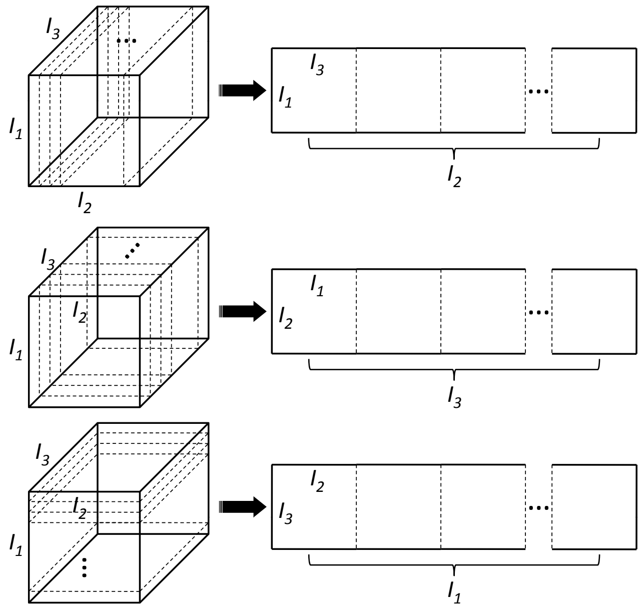


Figure 8: Unfolding the tensor $\mathcal{F} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ into three matrices, $F_1 \in \mathbb{R}^{I_1 \times I_2 I_3}$, $F_2 \in \mathbb{R}^{I_2 \times I_3 I_1}$, and $F_3 \in \mathbb{R}^{I_3 \times I_1 I_2}$

A Proofs

We first introduce the *matrix unfoldings* of a tensor, and then proceed to prove Theorem 1.

Definition 3. *The matrix unfoldings operation is the process of reordering the elements of a tensor into matrices. By performing "matrix unfoldings" on tensor $\mathcal{F} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, we obtain three matrices $F_1 \in \mathbb{R}^{I_1 \times I_2 I_3}$, $F_2 \in \mathbb{R}^{I_2 \times I_3 I_1}$, and $F_3 \in \mathbb{R}^{I_3 \times I_1 I_2}$. Figure 8 visualizes the unfolding processes. Intuitively, the matrix F_1 is formed by cutting the tensor $\mathcal{F}$ along the second dimension into several "matrix slices" and listing these matrices one after another. Please refer to [19, 20] for details of the operation.*

Theorem 1. *Let the Tucker decomposition of a third-order tensor $\mathcal{F} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ be denoted as*

$$\mathcal{F} \approx \hat{\mathcal{F}} = \mathcal{S} \times_1 Y^{(1)} \times_2 Y^{(2)} \times_3 Y^{(3)} \tag{25}$$

in which $Y^{(n)}$ for $n = 1, 2, 3$ are factor matrices, and $\mathcal{S} \in \mathbb{R}^{J_1 \times J_2 \times J_3}$ is a core tensor. Then the following holds.

$$||\hat{\mathcal{F}}_{:,t_i,:} - \hat{\mathcal{F}}_{:,t_j,:}||_F = \left((Y^{(2)}_{t_i,:} - Y^{(2)}_{t_j,:})\Sigma(Y^{(2)}_{t_i,:} - Y^{(2)}_{t_j,:})^T\right)^{\frac{1}{2}}$$

in which $\Sigma = S_2 S_2^T$ and S_2 is a matrix unfolding of the core tensor $\mathcal{S}$.

Proof. First, we prove the following equality

$$\mathrm{tr}\left(\hat{\mathcal{F}}_{:,t_i,:}\hat{\mathcal{F}}^T_{:,t_j,:}\right) = Y^{(2)}_{t_i,:}\Sigma(Y^{(2)})^T_{t_j,:}, \tag{26}$$

where $\mathrm{tr}(\cdot)$ denotes the trace of matrix.

From equation (25), we have

$$\begin{aligned}\hat{\mathcal{F}}_{:,t_i,:} &= \mathcal{S} \times_1 Y^{(1)} \times_2 Y^{(2)}_{t_i,:} \times_3 Y^{(3)} \\ &= (\mathcal{S} \times_2 Y^{(2)}_{t_i,:}) \times_1 Y^{(1)} \times_3 Y^{(3)}\end{aligned}$$

Moreover, by the definition of the n-mode product of tensors, the product $(\mathcal{S} \times_2 Y^{(2)}_{t_i,:})$ results in a matrix, and both $Y^{(1)}$ and $Y^{(3)}$ are matrices. It follows that

$$\hat{\mathcal{F}}_{:,t_i,:} = Y^{(1)} \times (\mathcal{S} \times_2 Y^{(2)}_{t_i,:}) \times (Y^{(3)})^T.$$

Similarly,

$$\hat{\mathcal{F}}_{:,t_j,:} = Y^{(1)} \times (\mathcal{S} \times_2 Y^{(2)}_{t_j,:}) \times (Y^{(3)})^T.$$

Since the columns of factor matrices $Y^{(n)}$ for $n = 1, 2, 3$ are orthogonal,

$$\begin{aligned}\operatorname{tr}\left(\hat{\mathcal{F}}_{:,t_i,:}\hat{\mathcal{F}}^T_{:,t_j,:}\right) &= \operatorname{tr}\left((\mathcal{S} \times_2 Y^{(2)}_{t_i,:})(\mathcal{S} \times_2 Y^{(2)}_{t_j,:})^T\right) \\ &= (Y^{(2)}_{t_i,:}S_2)(Y^{(2)}_{t_j,:}S_2)^T = Y^{(2)}_{t_i,:}\Sigma(Y^{(2)})^T_{t_j,:}.\end{aligned}$$

Equality (26) holds.

According to the definition of Frobenius-norm,

$$\begin{aligned}||\hat{\mathcal{F}}_{:,t_i,:} - \hat{\mathcal{F}}_{:,t_j,:}||_F &= \left(\operatorname{tr}\left((\hat{\mathcal{F}}_{:,t_i,:} - \hat{\mathcal{F}}_{:,t_j,:})(\hat{\mathcal{F}}_{:,t_i,:} - \hat{\mathcal{F}}_{:,t_j,:})^T\right)\right)^{\frac{1}{2}} \\ &= \left(\operatorname{tr}\left(\hat{\mathcal{F}}_{:,t_i,:}\hat{\mathcal{F}}^T_{:,t_i,:}\right) + \operatorname{tr}\left(\hat{\mathcal{F}}_{:,t_j,:}\hat{\mathcal{F}}^T_{:,t_j,:}\right) - 2\operatorname{tr}\left(\hat{\mathcal{F}}_{:,t_i,:}\hat{\mathcal{F}}^T_{:,t_j,:}\right)\right)^{\frac{1}{2}}\end{aligned}$$

together with the equation (26), we have

$$||\hat{\mathcal{F}}_{:,t_i,:} - \hat{\mathcal{F}}_{:,t_j,:}||_F = \left((Y^{(2)}_{t_i,:} - Y^{(2)}_{t_j,:})\Sigma(Y^{(2)}_{t_i,:} - Y^{(2)}_{t_j,:})^T\right)^{\frac{1}{2}}$$

□

Theorem 2. *The matrix $\Sigma = S_2S_2^T$ in Theorem 1 is diagonal and $\Sigma_{i,i}$ is the square of the ith singular value of matrix Z_2 which is a matrix unfolding of tensor $\mathcal{Z} = \mathcal{F} \times_1 (Y^{(1)})^T \times_3 (Y^{(3)})^T$*

Proof. From equation (16), we have

$$\mathcal{S}_{:,i,:} = (\mathcal{F} \times_1 (Y^{(1)})^T \times_3 (Y^{(3)})^T) \times_2 ((Y^{(2)})^T)_{i,:} = \mathcal{Z} \times_2 ((Y^{(2)})^T)_{i,:}$$

for $1 \leq i \leq J_2$.

Therefore,

$$\begin{aligned}\Sigma_{i,j} &= (S_2)_{i,:}(S_2)^T_{j,:} \\ &= \operatorname{tr}\left(\mathcal{S}_{:,i,:}\mathcal{S}^T_{:,j,:}\right) \\ &= \operatorname{tr}\left(\left(\mathcal{Z} \times_2 ((Y^{(2)})^T)_{i,:}\right)\left(\mathcal{Z} \times_2 ((Y^{(2)})^T)_{j,:}\right)^T\right) \\ &= ((Y^{(2)})^T)_{i,:}Z_2Z_2^T((Y^{(2)})^T)_{j,:}\end{aligned}$$

Since $Y^{(2)}$ is the most dominant J_2 singular vectors of Z_2, it follows that

$$\Sigma_{i,j} = \begin{cases} 0, & \text{if } i \neq j \\ \sigma_i^2, & \text{otherwise} \end{cases}$$

where σ_i is the ith singular value of matrix Z_2. □

B Algorithms

Algorithm 2 outlines the implementation of the *alternating least squares* (ALS) principle [20] for third-order tensors, which is invoked from Algorithm 1. The first step initializes factor matrices. In the second phase, based on the ALS principle [20], the factor matrices and the core tensor are iteratively adjusted until $\|\mathcal{S}\|_T$ ceases to increase. In the algorithm, the function call `SVD`(M) returns 3 matrices (U, V, W) that constitute a singular value decomposition of M with the property $M = UVW^T$. The function call `unfold`$(n, \mathcal{X})$ returns the matrix unfolding X_n of $\mathcal{X}$.

Algorithm 2: ALS

```
Input: tensor $\mathcal{F} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, target core tensor dimensions $J_1$, $J_2$, $J_3$
Output: core tensor $\mathcal{S}$, factor matrices $Y^{(1)}$, $Y^{(2)}$, $Y^{(3)}$, temporary matrix $\Lambda_2$

for $n = 2, 3$ do // Initialization
    $(U_n, \Lambda_n, V_n^T) \leftarrow$ SVD(unfold($n, F$))
    $Y^{(n)} \leftarrow$ most dominant $J_n$ singular vectors of $U_n$
end
repeat
    for $(i, j, k) = (1, 2, 3), (3, 1, 2), (2, 3, 1)$ do
        $\mathcal{Z} \leftarrow \mathcal{F} \times_j (Y^{(j)})^T \times_k (Y^{(k)})^T$
        $(U_i, \Lambda_i, V_i^T) \leftarrow$ SVD(unfold($i, Z$))
        $Y^{(i)} \leftarrow$ most dominant $J_i$ singular vectors of $U_i$
    end
    $\mathcal{S} \leftarrow \mathcal{Z} \times_2 (Y^{(2)})^T$
until $\|\mathcal{S}\|_T$ does not increase
return $(\mathcal{S}, Y^{(1)}, Y^{(2)}, Y^{(3)}, \Lambda_2)$
```

Table 9: Samples of search results produced by our approach

Queries	Top-5 Relevant URLs	ODP Categories
car, shopping	http://www.cars.com/	top/shopping/classifieds/automotive
		top/regional/north america/united states/illinois/localities/c/chicago/business and economy/automotive
	http://www.kbb.com/	top/home/consumer information/automobiles/purchasing
	http://www.nadaguides.com/	top/home/consumer information/automobiles/purchasing
	http://www.carbuyingtips.com/	top/home/consumer information/automobiles/purchasing
	http://www.autobytel.com/	top/shopping/vehicles/autos/buyer services
network, security	http://www.securityfocus.com/	top/computers/security/news and media
		top/computers/hacking/exploits
	https://grc.com/x/ne.dll?bh0bkyd2	top/computers/security/internet/services
	http://www.ethereal.com/	top/computers/software/operating systems/linux/projects/networking
		top/computers/software/operating systems/linux/projects/security
	http://www.bleedingsnort.com/	top/computers/security/intrusion detection systems/products and tools/open source
	http://www.nanog.org/	top/computers/internet/organizations/internet backbone
mp3, radio, music, online	http://nugs.net/	top/arts/music/styles/r/rock/jam/mp3
		top/arts/music/concerts and events/internet broadcasts
	http://www.soundclick.com/	top/arts/music/sound files/mp3
		top/arts/music/sound files/asf
		top/business/arts and entertainment/music/production/m
	http://www.radiolovers.com/	top/arts/radio/formats/old-time radio/streaming mp3 stations
	http://www.kurthanson.com/	top/computers/internet/broadcasting/information
	http://www.dxtuners.com/	top/recreation/radio/shortwave and dx listening
download, software, mp3	http://www.mediamonkey.com/	top/computers/multimedia/music and audio/audio formats/mp3/software/all-in-one
	http://www.audiograbber.com-us.net/	top/computers/multimedia/music and audio/audio formats/mpeg-4/software
		top/computers/multimedia/music and audio/audio formats/mp3/software/rippers
		top/computers/software/shareware/windows/multimedia/music and audio/mp3/rippers
	http://www.winamp.com/	top/computers/multimedia/music and audio/audio formats/mpeg-4/software
		top/computers/multimedia/music and audio/audio formats/mod/software
		top/computers/multimedia/music and audio/audio formats/mp3/software/players
		top/computers/multimedia/music and audio/audio formats/mp3/software/all-in-one
		top/computers/software/shareware/windows/multimedia/music and audio/players
	http://www.mp3000.net/	top/arts/music/sound files/mp3/search engines
	http://www.emp3world.com/	top/arts/music/sound files/mp3/search engines
		top/arts/music/sound files/mp3/downloads