ClustHOSVD: Item Recommendation by Combining Semantically-enhanced Tag Clustering with Tensor HOSVD

Panagiotis Symeonidis

Aristotle University, Department of Informatics, Thessaloniki, Greece
symeon@csd.auth.gr

ABSTRACT

Social Tagging Systems (STSs) allow users to annotate information items (songs, pictures, etc.) to provide them item/tag or even user recommendations. STSs consist of three main types of entities: users, items and tags. These data usually are represented by a 3-order tensor, on which Tucker Decomposition (TD) models are performed, such as Higher Order Singular Value Decomposition. However, TD models require cubic computations for the tensor decomposition. Furthermore, TD models suffer from sparsity that incurs in social tagging data. Thus, TD models have limited applicability to large-scale data sets, due to their computational complexity and data sparsity.

In this paper, we use two different ways to compute similarity/distance between tags (i.e., the TFIDF vector space model and the semantic similarity of tags using the ontology of Wordnet). Moreover, to reduce the size of the tensor's dimensions and its data sparsity, we use clustering methods (i.e., k-means, spectral clustering etc.) for discovering tag clusters, which are the intermediaries between a user's profile and items. Thus, instead of inserting the tag dimension in the tensor, we insert the tag cluster dimension, which is smaller and has less noise, resulting to better item recommendation accuracy. We perform experimental comparison of the proposed method against a state-of-the-art item recommendation algorithm with two real data sets (Last.fm and BibSonomy). Our results show significant improvements in terms of effectiveness and efficiency.

Categories and Subject Descriptors

H.3.3 [**Information Search-Retrieval**]: Information Filtering

1. INTRODUCTION

Social Tagging Systems (STSs) are web applications where users can upload, tag, and share information items (e.g., web videos, photos, etc.) with other users. Most STSs promote decentralization of content control and permit unsupervised tagging; users are free to use any tag they wish to describe an information item. Several Tucker Decomposition (TD) models [29], have been applied on 3-dimensional tensors to reveal the latent semantic associations between users, items and tags. However, the problem with TD models is the cubic core tensor, which results in a cubic runtime for the factorization dimension. Moreover, TD models are unfeasible for large-scale data sets, since these data sets have high factorization dimensions and many missing values.

In this paper, inspired by clustering-based methods [12, 26] which have been used in tag recommendation, we combine the tensor dimensionality reduction (i.e., tensor HOSVD) with the clustering of tags. We aim to find an effective preprocessing step, i.e., the clustering of tags, that can reduce the size of the tensor dimensions and deal with the missing values. To perform clustering of tags, we initially incorporate in our model two different auxiliary ways to compute the similarity/distance between tags. First, we compute the cosine similarity of tags based on the TFIDF weighting schema within the vector space model. Second, to address polysemy and synonymity of tags, we compute also their semantic similarity by utilizing the WordNet[1] dictionary.

After performing tag clustering, we use the centroids of the found tag clusters as representatives for the tensor tag dimension. As a result, we efficiently overcome the tensor's computational bottleneck by reducing both the factorization dimension and the data sparsity. Moreover, the clustering of tags is an effective means to reduce tag ambiguity and tag redundancy, resulting to better accuracy prediction and item recommendations. We have used three different clustering methods (i.e., k-means, spectral clustering and hierarchical agglomerative clustering) for discovering tag clusters. As will be later experimentally shown, spectral clustering outperforms the other methods, due to its flexibility to capture a wider range of cluster geometries and shapes [32].

The rest of this paper is organized as follows. Section 2 summarizes the related work. The intuition of the proposed approach is described in Section 3. The main parts of the ClustHOSVD algorithm are described in Section 4. Experimental results are given in Sections 5 and 6. In Section 7, we address several issues of our method and propose solutions. Finally, Section 8 concludes this paper.

2. RELATED WORK

A major drawback of Tucker Decompositions (TD) models such as HOSVD is that the construction of the core tensor requires cubic runtime in the factorization dimension for

[1]http://wordnet.princeton.edu

both prediction and learning. Moreover, they suffer from sparsity that incurs in social tagging data. To overcome the aforementioned problem, HOSVD can be performed efficiently following the approach of Sun and Kolda [18]. Other approaches to improve the scalability to large data sets is through slicing [30] or approximation [9]. Rendle et al. [23] proposed Ranking with Tensor Factorization (RTF), a method for learning an optimal factorization of a tensor for the specific problem of tag recommendations. Moreover, Steffen Rendle and Lars Schmidt-Thieme [24] proposed the Pairwise Interaction Tensor Factorization model (PITF), which is a special case of the Tucker Decomposition model with a linear runtime both for learning and prediction. PITF explicitly models the pairwise interactions between users, items and tags. It has been experimentally shown that PITF model outperforms Tucker Decomposition in terms of runtime. Notice that PITF [24] has won the ECML/PKDD Discovery Challenge 2009 for the accurate prediction task of graph-based tag recommendation. Cai et al. [4] proposed Low-Order Tensor Decompositions (LOTD), which also targets the very sparse data problem for tag recommendation. Their LOTD method is based on low-order polynomials that present low functional complexity. LOTD is able of enhancing statistics and avoids over-fitting, which is a problem of traditional tensor decompositions such as Tucker and Parafac decompositions. It has been experimentally shown [4] with extensive experiments on several data sets that LOTD outperforms PITF and other methods in terms of efficiency and accuracy. Another method which outperformed PITF is proposed by Gemmell et al. [11]. Their method builds a weighted hybrid tag recommender that blends multiple recommendation components drawing separately on complementary dimensions. Moreover, Leginus et al. [20], similarly to our work, improved tensor-based recommenders with clustering. They reduced the tag space by exploiting clustering techniques so that both the quality of recommendations and execution time are improved. However, they do not incorporate in their method the usage of TFIDF feature weighting scheme (i.e., intra-tag similarity and inter-tag dissimilarity), which can improve the accuracy of item recommendations. Finally, Rafailidis and Daras [22] proposed the TFC model, which is a Tensor Factorization and Tag Clustering model that uses a TFIDF weighting scheme. However, they did not incorporate additional information, from the semantic tags, in their model.

3. THE INTUITION OF THE PROPOSED APPROACH

In this section we present the intuition of our proposed approach and provide the outline of the approach through a motivating example, to grasp its essential characteristics. The main intuition of our method is based on the fact that if we perform Tag Clustering before the tensor construction, then we will be able to build a lower dimension tensor based on the found tag clusters.

3.1 Applying HOSVD on a tensor

When using a social tagging system, a user u tags an item i with a tag t, in order to be able to retrieve information items easily. Thus, the tagging system accumulates a collection of usage data, which can be represented by a set of triplets $\{u, t, i\}$. As illustrated in Figure 1, 3 users used 5 tags to annotate 3 different items. Notice that the sequence of arrows, which have the same label (annotation) between a user and an item, represents that a user used a tag for annotating the specific item. Thus, the annotated numbers on the arrow lines give the correspondence between the three types of objects. For example, user U_1 tagged item I_1 with tag "FIAT", denoted as T_1. The remaining tags are "BMW", denoted as T_2, "JAGUAR", denoted as T_3, "CAT" denoted as T_4, and "DOG" denoted as T_5. From Figure 1, we can infer that users U_1 and U_2 have common interests on cars, while user U_3 is interested in animals. A 3-order tensor $\mathcal{A} \in R^{3\times3\times3}$, can be constructed from the usage data. We use the co-occurrence frequency (denoted as weight) of each triplet user, item and tag as the elements of tensor $\mathcal{A}$, which are given in Table 1 and a 3-D visual representation of the tensor is shown in Figure 2. HOSVD is applied on the 3-order tensor constructed from these usage data. It uses as input the usage data of a tensor $\mathcal{A}$ and outputs the reconstructed tensor $\hat{\mathcal{A}}$. $\hat{\mathcal{A}}$ measures the associations among the users, items and tags. The elements of $\hat{\mathcal{A}}$ can be represented by a quadruplet $\{u, i, t, w\}$, where w measures the likeliness that user u will tag item i with tag t. Therefore, items can be recommended to u according to their weights associated with $\{u, t\}$ pair.

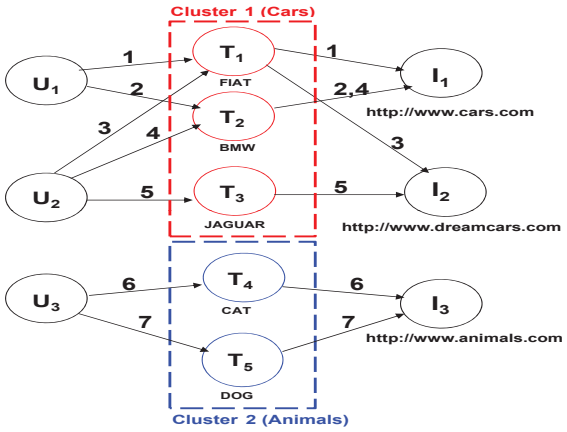


Figure 1: The Hypergraph of the running example

Table 1: Usage data of our running example.

Arrow Line	User	Tag	Item	Weight
1	U_1	T_1	I_1	1
2	U_1	T_2	I_1	1
3	U_2	T_1	I_2	1
4	U_2	T_2	I_1	1
5	U_2	T_3	I_2	1
6	U_3	T_4	I_3	1
7	U_3	T_5	I_3	1

3.2 Applying ClustHOSVD on a tensor

In this subsection, we show the intuition behind our approach, denoted as ClustHOSVD, to our running example. The main intuition is that, instead of inserting the tag dimension in the tensor, we can insert the tag cluster dimension, which is smaller and has less noise. As illustrated in Figure 1, 2 tag clusters have been found (Cluster 1 and Clus-

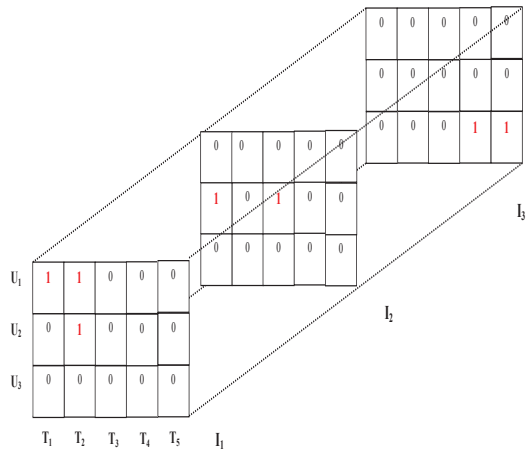


Figure 2: Visual representation of the initial tensor of our running example

ter 2). The first tag cluster C_1 refers to cars, whereas the second tag cluster C_2 refers to animals.

Based on the found tag clusters, we weight the interaction (i) between a user and a cluster centroid and (ii) between a tag and a cluster centroid. Then, we re-compute a weight for each (user, tag cluster, item) triplet as shown in Table 2, which is based on the distance of each tag from the tag cluster centroid. That is, we aggregate the distance of each tag vector (within the cluster) from its cluster centroid and divide it with the number of tags that belong in the same cluster. This weight will be used as input for the construction of our new initial tensor (for more details see Section 4.3.1).

Table 2: Usage Data of the running example after performing tag clustering.

Arrow Line	User	Tag Cluster	Item	Weight
1	U_1	C_1	I_1	0.7371
2	U_2	C_1	I_1	0.8848
3	U_2	C_1	I_2	0.5897
4	U_3	C_2	I_3	1

Based on Table 2, we can rebuild our initial hypergraph (see Figure 1) to the one shown in Figure 3. As shown, this hypergraph is now reduced in terms of the tags' dimension and has in replacement the two found tag clusters (C_1, C_2).

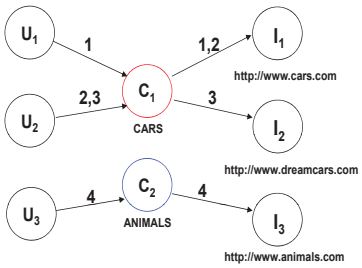


Figure 3: Transformed usage data of the running example after the tag clustering

The new ClustHOSVD tensor is shown in Figure 4.

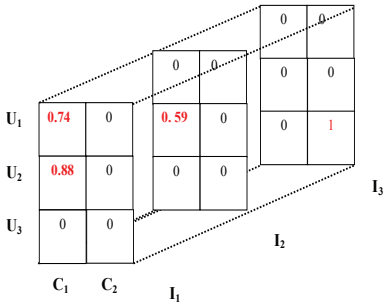


Figure 4: Visual representation of the tensor of our running example after tag clustering

4. THE CLUSTHOSVD METHOD

The clustHOSVD algorithm consists of 3 main parts. In this section we provide details on how we perform (i) the tag clustering, (ii) the tensor construction and its dimensionality reduction, and (iii) item recommendation based on the detected latent associations. To perform clustering of tags, we compute the cosine similarity of tags based on the TFIDF weighting schema within the vector space model. Moreover, to address polysemy and synonymity of tags, we compute also their semantic similarity. After performing tag clustering, we use the centroids of the found tag clusters as representatives for the tensor tag dimension. Then, we apply HOSVD to decompose and reconstruct the 3-order tensor revealing latent relationships among users, items and tag clusters. Finally, to recommend an item, we employ the new approximated values of the reconstructed tensor.

4.1 Tag clustering

A basic step of our approach is the creation of tag clusters that interconnect a user with an item (see Figure 3). For many clustering techniques, the similarity between tags must be first calculated. The cosine or pearson similarity between two tags may be calculated by treating each tag as a vector over the sets of users and items. From the usage data triplets (user, tag, item) of our running example (see Figure 1), we build a matrix representation $F \in R^{I_t \times I_u I_i}$, where $|I_t|$, $|I_u|$, $|I_i|$ are the numbers of tags, users and items, respectively. Table 3 shows matrix F of our running example, where each column can be considered as a feature f of the tag vector.

	$f_1 = U_1$	$f_2 = U_2$	$f_3 = U_3$	$f_4 = I_1$	$f_5 = I_2$	$f_6 = I_3$
T_1	1	1	0	1	1	0
T_2	1	1	0	2	0	0
T_3	0	1	0	0	1	0
T_4	0	0	1	0	0	1
T_5	0	0	1	0	0	1

Table 3: Each row in Matrix F represents the tag vector over the sets of users and items.

4.1.1 *Applying the TFIDF feature weighting scheme*

We will weight the column features of matrix F, in order to find (i) those features which better describe tag t and (ii) those features which better distinguish it from other tags. The first set of features provides quantification of intra-tag similarity, while the second set of features provides quantification of inter-tag dissimilarity. Our model is mo-

tivated from the information retrieval field and the TFIDF scheme [1].

4.1.2 The Cosine Similarity measure

To perform tag clustering, we need to find the distances between tags. In our model, as it is expressed by Equation 1, we apply cosine similarity in the weighted tag-feature W matrix. We have selected cosine similarity because it computes the angle between the two tag vectors, which is insensitive to the absolute length of the tag vector. We adapt cosine similarity to take into account only the set of features, that are correlated with both tags [1]. Thus, in our model the similarity between two tags, t and s, is measured as follows:

$$\mathrm{CosSim}(t,s) = \frac{\sum_{\forall f \in I_u I_i} (w_{t,f} \cdot w_{s,f})}{\sqrt{\sum_{\forall f \in I_u I_i} (w_{t,f})^2} \sqrt{\sum_{\forall f \in I_u I_i} (w_{s,f})^2}} \quad (1)$$

4.1.3 The Semantic Similarity measure

Since now, we have performed the TFIDF scheme [1] to compute similarity among tags of W matrix based on the Vector Space Model, also known as the bag of words model. However, the lack of common features between two tags in W matrix does not necessarily mean that the tags are unrelated. Similarly, relevant tags may not contain the same features in W matrix. For example, the TFIDF scheme [1] and the Vector Space Model cannot recognize synonyms or semantically similar terms (e.g., "car", "automobile"). The process could be improved by also incorporating ontology-based semantic similarities between tags. Ontologies can be seen as a graph in which concepts are the nodes, which are interrelated mainly by means of taxonomic (is-a) edges.

To measure the semantic similarity among tags, we can use several ontology-based similarity measures, which are classified to the following three categories[25]: (i) Edge-counting (ii) Feature-based (iii) Information Content-based. Edge-counting approaches calculate similarity between concepts (nodes) by computing the minimum path length connecting their corresponding nodes via (is-a) links. The longer the path, the more semantically far the tags are. In our approach, we use an edge-counting approach because it is simple, easily interpretable and its evaluation requires a low computation cost. However, the effectiveness of the edge-counting approach relies on the fact that all links in the taxonomy must represent a uniform distance. In practice, the semantic distance among tag specializations/generalizations in an ontology would depend on the size of the taxonomy, the degree of granularity and taxonomic detail implemented by the knowledge engineer[25].

In our approach, we compute similarity of tags based on their semantic affinity in WordNet, where parts of speech words (i.e, noun, verb, etc.) are categorized into taxonomies and each node is a set of synonyms (synset). If a tag has more than one meanings (polysemy), it will appear in multiple synsets at various locations in the taxonomy. To measure the semantic similarity between two tags, t and s, we treat taxonomy as an undirected graph and measure their semantic similarity as follows [31]:

$$\mathrm{SemSim}(t,s) = \frac{2 \cdot depth(LCS)}{N1 + N2 + 2 \cdot depth(LCS)} \quad (2)$$

where LCS is the least common super-concept of t and s. $Depth(LCS)$ is the number of nodes to the path from LCS to root of the taxonomy. $N1$ is the number of nodes on the path from t to LCS. $N2$ is the number of nodes on the path from s to LCS. Notice that based on the above Equation, we assign higher similarity to tags which are close together and lower in the hierarchy. That is, if a tag is lower in the hierarchy it is assumed as more specific than other tags which are higher in the hierarchy and are assumed as more general.

4.1.4 Hybrid Similarity measure

In this Section, we combine linearly the classical cosine similarity (CosSim), as described in Section 4.1.2, with the semantic similarity (SemSim), which is explained in detail in Section 4.1.3 into a single similarity matrix, since both similarity matrices can contain valuable information. Thus, in our model the final similarity between two tags, t and s, is measured as shown by Equation 3:

$$\mathrm{Sim}(t,s) = (1-a) \cdot \mathrm{CosSim} + a \cdot \mathrm{SemSim} \quad (3)$$

In Equation 3, a takes values between [0,1]. This parameter can be adjusted by the user. When a takes values greater than 0.5, then semantic similarity values matrix have much more impact in the final similarity values than the similarity values based on the W matrix. When a becomes zero, the final similarity values are exactly the similarity values based on W matrix only. When a becomes one, the final similarity values are exactly the similarity values based on semantic similarity.

Please notice that in several cases the distribution of the similarity values in the interval [0,1] between CosSim and SemSim differ significantly. For example, consider the case that the most similarity values in CosSim are normally distributed between 0 and 0.3, whereas the most similarity values in SemSim are normally distributed between 0.4 and 0.7. Then, it is unfair to take a simple weighted average of them using Equation 3, because the similarity values of CosSim will always be dominated by those of SemSim. To address this problem, we can apply either min-max scaling or standardization.

An alternative way to fuse the aforementioned similarity matrices is to consider learning an individual kernel matrix K_i from each different similarity matrix (i.e., CosSim and SemSim), and invoke a Multiple Kernel Learning (MKL) setup to obtain a final kernel matrix K. That is, for each similarity matrix we have to non-linearly map its contents to a higher dimensional space using a mapping function. However, to avoid the explicit computation of the mapping function, all computations should be done in the form of inner products. Then, we can substitute the computation of inner products with the results of a kernel function. This technique is called the "kernel trick" [6] and avoids the explicit (and expensive) computation of each individual kernel matrix K_i. Regarding the kernel functions, in our experiments we use the Gaussian kernel $K(t,s) = e^{-\frac{||t-s||^2}{2c^2}}$, and the Exponential Kernel $K(t,s) = e^{-\frac{||t-s||}{2c^2}}$, which are commonly used in many applications. As parameter c, we use the estimate for standard deviation of each similarity matrix. Finally, we combine the kernel matrices by averaging their resulted values based on the averaging MKL function.

4.2 Algorithms for Tag Clustering

In this Section, we present three clustering techniques: hierarchical agglomerative clustering, k-means clustering, and multi-way spectral clustering. We have chosen the first technique because it offers different levels of clustering granularity, which can provide different application-specific insights. Moreover, we have chosen k-means because it belongs to the representative-based algorithms and it is the simplest of all clustering algorithms, since it relies on the intuitive notion of similarity among cluster data points. Finally, we used as comparison partner the multi-way spectral clustering, which belongs to the graph-based family of clustering algorithms, because it can capture different sized clusters and convex shapes, whereas k-means can capture mainly globular cluster shapes.

4.2.1 Hierarchical agglomerative clustering

Hierarchical agglomerative clustering [27] starts with the tags as individual clusters and, at each step, merges the closest pair of clusters until only one tag cluster remains. This requires the notion of tag cluster proximity. In this paper, we use group average measure, where the proximity between two clusters is defined as the average pairwise proximity among all pairs of tags in the different clusters. In our running example, the proximity between tags is captured with distance (dissimilarity) matrix T.

The input parameters are similarity step and cut off coefficient. The parameter similarity step is the decrement by which the similarity threshold is lowered and controls the granularity of the derived clusters. An optimal value for similarity step would aggregate tags slowly enough to capture the relationship between the concepts of individual clusters. Aggregating the tags too quickly may over-specify the structure and lose important tag interdependencies. The cut off coefficient controls the level where the hierarchy of tags is dissected into individual clusters. If the cut off coefficient is set too high, the result will be many small clusters, failing to capture possible tag redundancies. In contrast, if the cut off coefficient is set too low, the result is few big clusters with low cohesiveness and high concept ambiguity.

4.2.2 k-means

Given a set of tags (T_1, T_2, ...,T_n), k-means [21] aims to partition the n tags into k sets ($k < n$) C=(C_1, C_2,..., C_k) so as to minimize the within-cluster sum of squared error(SSE), as shown by Equation 4:

$$SSE = \sum_{i=1}^{k} \sum_{T_x \in C_i} dist(T_x, c_i)^2, \quad (4)$$

where T_x is a tag in cluster C_i and c_i is the centroid point for cluster C_i. k-means chooses k initial centroids, where k is a user-specified parameter, namely, the number of clusters desired. Each node is then assigned to the closest centroid, and each collection of tags assigned to a centroid is a cluster. The centroid of each cluster is then updated based on the tags assigned to the cluster. This procedure is repeated until no tag changes cluster, or equivalently, until the centroids remain the same. Please notice that we use a special version of k-means algorithm using the cosine similarity, known as spherical k-means algorithm [7], where $dist(T_x, c_i) = 1 - CosSim(T_x, c_i)$. We have selected cosine similarity because it computes the angle between the two tag vectors, which is insensitive to the absolute length of the tag vector.

4.2.3 multi-way Spectral Clustering

Spectral algorithms are based on eigenvectors of Laplacians, which are a combination of the distance and the degree matrix. The normalized laplacian matrix is computed by Equation: $\mathcal{L} = \mathcal{D}^{-\frac{1}{2}} \times (\mathcal{D} - T) \times \mathcal{D}^{-\frac{1}{2}}$, where $\mathcal{D}$, T are the degree, and the distance/dissimilarity tag matrices, respectively. Please notice that the degree matrix is a diagonal matrix which contains information about the degree of each vertex (i.e., the number of edges attached to each vertex).

Multi-way spectral clustering algorithm [17, 28] computes the first k eigenvectors $e_1, \ldots, e_k$ with the corresponding $\lambda_1, \ldots, \lambda_k$ eigenvalues of the normalized laplacian matrix L based on Equation $L \times U = \lambda \times U$. The U matrix has columns the eigenvectors $e_1, \ldots, e_k$, whereas tags $T_i \in T$, with $i = 1, \ldots, n$, correspond to the i-row of U. In the beginning, we compute the first k eigenvectors and the first λ eigenvalues of the normalized laplacian matrix L. Next, we cluster tags T_i with the k-means algorithm into clusters $C_1, \ldots, C_k$, based on the aforementioned eigenvectors e_k. Having defined the clusters we can now compute the centroids of each cluster and then compute the distances of each tag from each cluster centroid.

4.3 HOSVD for Tensor Reduction

Our Tensor Reduction approach initially constructs a tensor $\mathcal{A}$, based on usage data triplets $\{u, c, i\}$ of users, tag clusters, items. Consequently, we proceed to the unfolding of $\mathcal{A}$, where we build three new matrices. Then, we apply SVD in each new matrix. Finally, we build the core tensor $\mathcal{S}$ and the resulting tensor $\hat{\mathcal{A}}$. The 6 steps are described in the following subsections.

4.3.1 The initial construction of tensor $\mathcal{A}$

We transform the initial usage data triplets (user, tag, item) to (user, tag clusters, item) triplets. To do this, we treat each tag as a vector over the sets of users and items. Each dimension of the tag vector corresponds to a separate user or item. The numerical representation of a tag vector is defined as follows: The value in a dimension of the tag vector is based on either (i) the frequency that a user used this particular tag for annotating an item or (ii) the frequency that an item is annotated with this tag by all users. Then, the centroid of a tag cluster is computed as the average vector of the tags' vectors, which belong in the cluster. In our running example, Table 3 shows matrix F, where each row presents a tag's vector. Since tags T_1, T_2, and T_3 belong to the first tag cluster C_1, the cluster centroid of C_1, takes values (0.666, 1, 0, 1, 0.333 and 0).

As a next step, we re-compute a weight for each (user, tag cluster, item) triplet as shown in Table 2, which is based on the distance of each tag from the tag cluster centroid. For example, if we want to capture only the cosine similarity between tags (not semantic similarity), we can set the value of parameter α of Equation 3 equal to 0. Then, we can compute the first value (e.g., 0.74) of Table 2, by calculating the cosine similarity between the C_1 cluster centroid vector (i.e, 0.666, 1, 0, 1, 0.333 and 0) and the vector of user U_1 (i.e., 0.666, 0, 0, 0,666, 0, 0). This weight will be used as input for the construction our new tensor. This new tensor is smaller in terms of the tag dimension.

From the usage data triplets (user, tag clusters, item), we construct an initial 3-order tensor $\mathcal{A} \in R^{I_u \times I_i \times I_c}$, where $|I_u|$, $|I_i|$, $|I_c|$ are the numbers of users, items and tag clusters, respectively. Each tensor element measures the number of times that a user u tagged item i with a tag which belong in tag cluster c.

4.3.2 Matrix unfolding of tensor $\mathcal{A}$

Tensor $\mathcal{A}$ can be unfolded (matricized), i.e., we build matrix representations of tensor $\mathcal{A}$ in which all the column (row) vectors are stacked one after the other. The initial tensor $\mathcal{A}$ is matricized in all three modes. Thus, after the unfolding of tensor $\mathcal{A}$ for all three modes, we create 3 new matrices A_1, A_2, A_3, as follows:

$$A_1 \in R^{I_u \times I_i I_c},$$
$$A_2 \in R^{I_i \times I_u I_c},$$
$$A_3 \in R^{I_u I_i \times I_c}$$

4.3.3 Application of SVD on each matrix

We apply SVD on the three matrix unfoldings A_1, A_2, A_3. We result, in total, to 9 new matrices.

$$A_1 = U^{(1)} \cdot S_1 \cdot V_1^T \quad (5)$$
$$A_2 = U^{(2)} \cdot S_2 \cdot V_2^T \quad (6)$$
$$A_3 = U^{(3)} \cdot S_3 \cdot V_3^T \quad (7)$$

For Tensor Dimensionality Reduction, there are three dimensional parameters to be determined. The numbers p_1, p_2 and p_3 of left singular vectors of matrices $U^{(1)}$, $U^{(2)}$, $U^{(3)}$, respectively, that are preserved. They will determine the final dimensionality of the core tensor S. Since each of the three diagonal singular matrices S_1, S_2 and S_3 are calculated by applying SVD on matrices A_1, A_2 and A_3, respectively, we use different p_1, p_2 and p_3 numbers of principal components for each matrix $U^{(1)}$, $U^{(2)}$, $U^{(3)}$. The numbers p_1, p_2 and p_3 of singular vectors are chosen by preserving a percentage of information of the original S_1, S_2, S_3 matrices after appropriate tuning (the default percentage is set to 50% of the original matrix).

4.3.4 The construction of the core tensor $\mathcal{S}$

The core tensor $\mathcal{S}$ governs the interactions among users, items and tags. Since we have selected the dimensions of $U^{(1)}$, $U^{(2)}$ and $U^{(3)}$ matrices, we proceed to the construction of $\mathcal{S}$, as follows:

$$\mathcal{S} = \mathcal{A} \times_1 {U_{p_1}}^{(1)^T} \times_2 {U_{p_2}}^{(2)^T} \times_3 {U_{p_3}}^{(3)^T}, \quad (8)$$

where $\mathcal{A}$ is the initial tensor, ${U_{p_1}}^{(1)^T}$ is the tranpose of the p_1-dimensionally reduced $U^{(1)}$ matrix, ${U_{p_2}}^{(2)^T}$ is the transpose of the p_2-dimensionally reduced $U^{(2)}$ matrix, ${U_{p_3}}^{(3)^T}$ is the tranpose of the p_3-dimensionally reduced $U^{(3)}$ matrix.

4.3.5 The re-construction of tensor $\hat{\mathcal{A}}$

Finally, tensor $\hat{\mathcal{A}}$ is build as the product of the core tensor $\mathcal{S}$ and the mode products of the three matrices $U^{(1)}$, $U^{(2)}$ and $U^{(3)}$ as follows:

$$\hat{\mathcal{A}} = \mathcal{S} \times_1 {U_{p_1}}^{(1)} \times_2 {U_{p_2}}^{(2)} \times_3 {U_{p_3}}^{(3)}, \quad (9)$$

$\mathcal{S}$ is the reduced core tensor, ${U_{p_1}}^{(1)}$ is the p_1-dimensionally reduced $U^{(1)}$ matrix, ${U_{p_2}}^{(2)}$ is the p_2-dimensionally reduced $U^{(2)}$ matrix, ${U_{p_3}}^{(3)}$ is the p_3-dimensionally reduced $U^{(3)}$ matrix.

4.3.6 Alternating Least Squares Algorithm for Fitting the Tensor

The reconstructed tensor of the previous subsection (also known as truncated HOSVD) is not optimal, but is a good starting point for an iterative alternating least squares (ALS) algorithm to best fit the reconstructed tensor to the original one [19]. The basic idea of AlsHOSVD algorithm tries to minimize the error between the initial and the predicted values of the tensor. The pseudo-code of the approach is depicted in Algorithm 1.

Algorithm 1 AlsHOSVD

Input: The initial tensor $\mathcal{A}$ with user, tag cluster, and item dimensions.
Output: The approximate tensor $\hat{\mathcal{A}}$ with p_1, p_2 and p_3 left leading eigenvectors of each dimension, respectively.

1: Initialize core tensor $\mathcal{S}$ and left singular vectors ${U^{(1)}}, {U^{(2)}}, {U^{(3)}}$ of A_1, A_2, and A_3, respectively.
2: **repeat**
3: $\quad \mathcal{S} = \mathcal{A} \times_1 {U_{p_1}}^{(1)^T} \times_2 {U_{p_2}}^{(2)^T} \times_3 {U_{p_3}}^{(3)^T}$
4: $\quad \hat{\mathcal{A}} = \mathcal{S} \times_1 {U_{p_1}}^{(1)} \times_2 {U_{p_2}}^{(2)} \times_3 {U_{p_3}}^{(3)}$
5: $\quad {U_{p_1}}^{(1)} \leftarrow p_1$ leading left singular vectors of A_1
6: $\quad {U_{p_2}}^{(2)} \leftarrow p_2$ leading left singular vectors of A_2
7: $\quad {U_{p_3}}^{(3)} \leftarrow p_3$ leading left singular vectors of A_3
8: **until** $\|\mathcal{A} - \hat{\mathcal{A}}\|^2$ ceases to improve **OR** maximum iterations reached
9: **return** $\mathcal{S}$, ${U_{p_1}}^{(1)}$, ${U_{p_2}}^{(2)}$, and ${U_{p_3}}^{(3)}$

As shown in line 8 of Algorithm 1, AlsHOSVD minimizes an objective function that computes the error among real and predicted values of the original and the approximate tensors. This is done cyclically until our objective function ceases to fit to the original values or maximum user-defined iterations reached. Please notice, that the values of the leading left singular vectors in all 3-modes (lines 5-7) increased gradually in each repetition.

4.4 The Generation of the Item Recommendation list

In this paper, we assume that the user interacts with the system by selecting a query tag and expects to receive an item recommendation. To recommend an item, we can use calculated values of the processed tensor $\hat{\mathcal{A}}$. Tensor $\hat{\mathcal{A}}$ measures the latent associations among the users, items and tag clusters and acts as a model that is used during the item recommendation.

Each element of $\hat{\mathcal{A}}$ represents a quadruplet $\{u, c, i, w\}$, where w is the likeliness (i.e., also mentioned in previous sections as weight) that user u will tag item i with a tag from cluster c. Before tensor decomposition, the ternary relation of users, items and tag clusters can be depicted by the binary tensor $\mathcal{A} = (a_{u,c,i}) \in \mathbb{R}^{|U| \times |C| \times |I|}$ where 1 indicates observed tag assignments and 0 missing values (see Table 1). However, after the tensor decomposition l depicts the probability of the existence of a ternary relation, denoted also in this paper as weight (see Table 2). Therefore, for a user u and a tag t, items can be recommended according to their weights associated with $\{u, c\}$ pair. If we want to recommend to u N items for tag t which belongs in tag cluster c, then we select the N corresponding items with the highest weights.

4.5 The ClustHOSVD Algorithm and its complexity

Figure 5 depicts the outline of ClustHOSVD. The input is the initial usage data triplets (user, tag, item), a selected user u and a tag t that u is interested for. The output is the reduced approximate tensor which incorporates the tag cluster dimension and a set of recommended items to user u.

Algorithm 2 ClustHOSVD
Input
n {user, tag, item}-triplets of the training data.
k: number of clusters
u: a selected user.
t: a selected tag.

Output
$\hat{\mathcal{A}}$: an approximate tensor with user, tag cluster, and item dimension.
N: the number of recommended items.

Step 1. Perform clustering (k-means, spectral etc.) on tag dimension.
1.a) Compute the tag-tag similarities and the k cluster centroids.
1.b) Compute the distance of each tag from the cluster centroid.
1.c) Execute the clustering on tag dimension.
Step 2. Apply HOSVD on Tensor.
2.a) Build the initial $\mathcal{A}$ tensor inserting the tag cluster dimension.
2.b) Perform HOSVD to de-compose and re-compose the $\mathcal{A}$ tensor. (steps 1-6 of Section 4.3).
Step 3. Generate the item recommendation list.
3.a) Get from the approximate tensor $\hat{\mathcal{A}}$ the w likeliness that user u will tag item i with a tag from cluster c.
3.b) Recommend the top-N items with the highest w likeliness to user u for tag t.

Figure 5: Outline of the ClustHOSVD Algorithm.

In step 1, the complexity of ClustHOSVD depends on the selected clustering algorithm. In case we apply k-means, its time complexity is O($I_c \times k \times i \times f$), where $|I_c|$ is the number of tag clusters, k is the number of clusters, i is the number of iterations until k-means converge, and f is the number of tag features, where each tag can be expressed as a f-dimensional vector. In case we apply multi-way spectral clustering, we can apply firstly k-means to cluster the tags of the tripartite graph and then we can apply spectral clustering only on the cluster centroids (representative tags of each cluster). By using this implementation, the overall computation cost of multi-way spectral clustering is O(k^3)+ O($I_c \times k \times i \times f$). Finally, the time complexity of hierarchical agglomerative clustering takes O(t^3) operations, where $|t|$ is the number of tags, which makes it slow for large data sets.

In step 2, the runtime complexity of HOSVD is cubic in the size of the latent dimensions. However, our ClustHOSVD performs clustering on the tag dimension, resulting usually to a small number of tag clusters. Notice that the same procedure can be followed for the other two dimensions (users and items). Thus, we can result to a tensor with a very small number of latent dimensions.

In step 3, the top-N recommended items are found after sorting the w likeliness values that user u will tag item i with a tag from cluster c, by using a sorting algorithm (quicksort) with complexity $O(I_i \log I_i)$, where $|I_i|$ is the number of items.

5. EXPERIMENTAL RESULTS

In this section, we present experimental results for the performance of our approach. Henceforth, our proposed approach is denoted as ClustHOSVD. There are three versions of ClustHOSVD. One is hierarchical agglomerative clustering combined with HOSVD, denoted as ClustHOSVD(hac), the second is k-means clustering combined with HOSVD, denoted as ClustHOSVD(k-means) and the third is spectral clustering combined with HOSVD, which is denoted as ClustHOSVD(spectral). Moreover, we include in our experiments the simple version of tensor HOSVD approach.

5.1 Data Sets

To evaluate the examined algorithms, we have chosen real data sets from two different social tagging systems: BibSonomy and Last.fm, which have been used as benchmarks in past works [4, 11, 15, 24].

Following the approach of Hotho et al. [15] to get more dense data, we adapt the notion of a p-core to tri-partite hypergraphs. The p-core of level p has the property that each user, tag and item has/occurs in at least p posts. For both data sets we used p = 5. Thus, for the Bibsonomy data set there are 105 users, 246 items and 591 tags, whereas for the Last.fm data set there are 112 users, 234 items and 567 tags.

5.2 Experimental Configuration

We performed 4-fold cross validation, thus each time we divide the data set into a training set and a test set with sizes 75% and 25% of the original set, respectively. All algorithms had the task to predict the items of the users' postings in the test set. All presented measurements (i.e., the average of the reported values between the runs), based on two-tailed t-test, are statistically significant at the 0.05 level. Please notice that we have run each experiment 50 times using different users each time.

Based on the approaches of Herlocker et al. and Huang et al. [13, 16], we consider the division of items of each test user into two sets: (i) the *past* items of the test user and, (ii) the *future* items of the test user. Therefore, for a test user we generate the item recommendations based only on the items in his past set.As performance measures we use the classic metrics of precision and recall.

5.3 Algorithms' Settings and Sensitivity analysis of the Core Tensor Dimensions

For each algorithm under comparison, we will now describe briefly the specific settings used to run it:

ClustHOSVD(hac): As described in Section 4.2.1, the input parameters for agglomerative hierarchical clustering (hac) are parameters similarity step and cut off coefficient. For each data set, we found the best tuning parameters. The best value of parameter similarity step was found to be 0.004 and 0.003 for BibSonomy and Last.fm data set, respectively. The best value for the cut off coefficient is 0.2 for the BibSonomy data set and 0.3 for the Last.fm data set. Based on the above parameters settings, the number of found tag clusters are 25 and 21 for Bibsonomy and Last.fm data sets, respectively. Thus, the tag cluster dimension of the initial tensor is 25 and 21 for Bibsonomy and Last.fm data sets, respectively.

Next, for the Bibsonomy data set, we study the influence of the core tensor dimensions on the accuracy performance of ClustHOSVD(hac). If one dimension of the core tensor is fixed, we found that the recommendation accuracy varies as the other two dimensions change, as shown in Figure 6. The vertical axes denote the precision and the other two axes denote the corresponding dimensions. The results concern the preci-

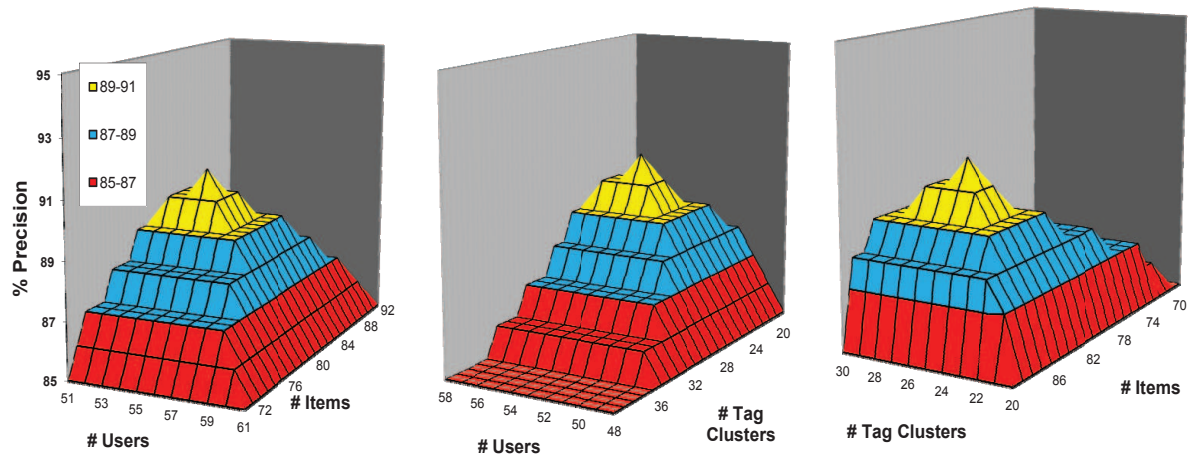


Figure 6: Performance of the Tensor Reduction algorithm as the dimensions of the core tensor vary for the BibSonomy data set. For the leftmost figure, the tag cluster dimension is fixed at 25 and the other two dimensions change. For the middle figure, the item dimension is fixed at 85. For the rightmost figure, the user dimension is fixed at 52.

sion found for the test set. For each figure, two dimensions are varied and one dimension is fixed. Thus, for the leftmost figure, the tag cluster dimension is fixed at 25 and the other two dimensions change. For the middle figure, the item dimension is fixed at 85. For the rightmost figure, the user dimension is fixed at 52.

Our experimental results indicate that a 50% of the original diagonal of S_1, S_2, S_3 matrices can give good approximations of A_1, A_2 and A_3 matrices. Thus, the numbers p_1, p_2, and p_3 of left singular vectors of matrices $U^{(1)}$, $U^{(2)}$, $U^{(3)}$ after appropriate tuning are set to 52, 85 and 25 for the BibSonomy data set, and are set to 28, 65 and 21 for the Last.fm data set. As far as the tuning is concern, it is done recursively. That is, we set each time specific values for the two dimensions and we compute a value for the third (i.e., p_1, p_2, and p_3) till there is no more improvement in the difference of element-wise values between the original and the approximated tensor. Please notice, that due to lack of space we do not present experiments for the tuning of p_1, p_2 and p_3 parameters for the other algorithms.

ClustHOSVD(k-means): In Section 4.2.2, one of the required input values for the k-means algorithm is the number k of clusters. To improve our recommendations in terms of effectiveness, it is important to fine-tune the k variable. The best k (number of clusters) was found to be 20 and 18 for the BibSonomy and Last.fm data sets, respectively. Please notice that for finding the number k of clusters to run ClustHOSVD (k-means), we have adopted the parameter tuning using an internal measure, i.e., the ratio of the intra-cluster to inter-cluster distance. That is, the variation in the intra-cluster to the inter-cluster distance ratio shows an inflection point (where the measure starts increasing instead of reducing) at the correct choice of the k parameter.

ClustHOSVD(spectral): Multi-way spectral clustering also requires a user-defined number k of clusters as an input parameter. The best k (number of clusters) was found to be 11 and 8 for the BibSonomy and Last.fm data sets, respectively. Please note that Zelnik-Manor and Perona [33] have proposed a method which automatically determines the number of clusters. They suggested exploiting the structure of the eigenvectors to infer automatically the number of clusters. In their approach, they define a cost function and try to minimise it by rotating the top few normalised eigenvectors of the laplacian matrix. The number of clusters is taken as the one providing the minimal cost in the objective function.

HOSVD: The numbers p_1, p_2 and p_3 of left singular vectors of matrices $U^{(1)}$, $U^{(2)}$, $U^{(3)}$ after appropriate tuning are set to 60, 105 and 225 for the BibSonomy data set, whereas are set to 40, 80 and 190 for the Last.fm data set.

Moreover, we test the performance of different aggregation functions (i.e. linear, Gausian kernel, and Exponential kernel) for the combination of similarity matrices, as described in Section 4.1.4. In particular, in Section 4.1.4, we presented the definition of our linear combined similarity measure (see Equation 3). In this section, we test different aggregation functions in order to discover the best precision values that we can attain when we recommend a top item to a user. The aggregation functions' performance can be seen in Figure 7 for the Bibsonomy and LastFM data sets, respectively. As shown, the best performance in all data sets is attained by the linear function, which indicates that appropriate tuning of parameter a could further leverage the precision performance.

For ClustHOSVD(spectal) algorithm, which - as will be shown later - attains the best results, we have varied the weights of a parameter in the similarity measure of Eq. 3, i.e., $a = 1.0$ (considering only semantic similarity) and $a = 0.99, 0.98, ..., 0.01, 0$ (gradual increase of the contribution of cosine similarity which is based on the Vector Space Model). For the Bibsonomy data set, the best performance is achieved by $a = 0.35$, i.e., when both the semantic information and the conventional cosine similarity on tensor un-

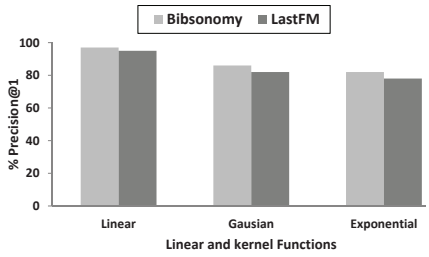


Figure 7: Precision comparison of linear, kernel Gausian, and kernel Exponential aggregation functions for the Bibsonomy and LastFM data sets.

folding are incorporated into the hybrid similarity function. For the Last.fm data set, the best performance is attained when a parameter is equal to 0.29. Henceforth, these values of parameter a will be used for the rest of the experiments.

5.4 Accuracy comparison of ClustHOSVD variations

In this section, we proceed with the comparison of HOSVD with clustHOSVD(hac), ClustHOSVD(k-means) and the ClustHOSVD(spectral), in terms of precision and recall. This reveals the robustness of each algorithm in attaining high recall with minimal losses in terms of precision. We examine the top-N ranked list of items, which is recommended to a test user, starting from the top item. In this situation, the recall and precision vary as we proceed with the examination of the top-N list.

For the BibSonomy data set (N is between [1..5]), in Figure 8, we plot a precision versus recall curve for all four algorithms. Each curve consists of five points, which is represented with squares, triangles, etc. The leftmost point of each curve represents the precision and recall values acquired when we recommend a top-1 item, whereas the rightmost point denotes the values we get for the top-5 recommended item. As shown, the precision of each algorithm falls as N increases. In contrast, as N increases, recall for all four algorithms increases too. As expected, spectral clustering outperforms hac and k-means, because it captures a wider range of cluster geometries and shapes. That is, k-means captures mainly globular cluster shapes, whereas hac cannot handle different sized clusters and convex shapes. Hac outperforms k-means because it is more flexible through parameters similarity step and cut off coefficient, which are not offered by k-means. That is, k-means puts almost every tag in a cluster. In contrast, hac uses similarity step to decide when to determine a tag in a cluster, resulting to more homogeneous clusters. As also shown, all ClustHOSVD methods outperform HOSVD. The reason is that clustering reduces noise, tag redundancy and tag ambiguity. These results clearly demonstrate that there is a significant improvement to be gained by using clustering before performing HOSVD.

For the Last.fm data set (N is between [1..5]), in Figure 9, we also plot a precision versus recall curve for all three algorithms. As shown, results are in accordance with those of the Bibsonomy Data set.

5.5 Time Comparison of ClustHOSVD variations with HOSVD

In this section, we proceed with the comparison of HOSVD

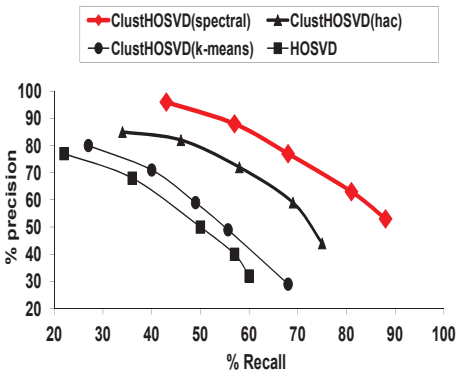


Figure 8: Comparison of HOSVD, clustHOSVD(k-mean) and clustHOSVD(spectral) algorithms for the BibSonomy dataset.

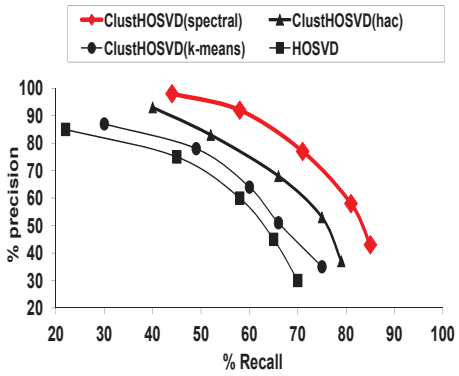


Figure 9: Comparison of HOSVD, clustHOSVD(k-mean) and clustHOSVD(spectral) algorithms for the Last.fm dataset.

with clustHOSVD(hac), ClustHOSVD(k-means) and the ClustHOSVD(spectral), in terms of efficiency using both real (Bibsonomy and Last.fm) and a synthetic data set. Each dimension {i.e., user, tag, item} of the synthetic data set is set to 10^4. That is, for the synthetic data set, we have created randomly posts which incorporate 10000 unique objects for each dimension, whereas each object appears in average 10 posts. We measured the clock time for the off-line parts of the algorithms (i.e., step 1 and 2 of our algorithm in Figure 5). The off-line part refers to the tag clustering and the calculation of the approximate tensor. We have performed a 50% reduce on the tensor dimension for all algorithms. The results are presented in Table 4.

Algorithm	Synthetic $\{10^4,10^4,10^4\}$	Bibsonomy {105,246,591}	Last.fm {112,234,567}
HOSVD(hac)	217 sec	8.4 sec	7.9 sec
HOSVD(k-means)	145.7 sec	6.45 sec	6.15 sec
HOSVD(spectral)	153.2 sec	9.1 sec	7.7 sec
HOSVD	352.4 sec	13.4 sec	12.6 sec

Table 4: Time comparison of all tested algorithms for a synthetic and two real data sets.

As shown, clustering of tag space improves the time performance of the factorization process and reduces the memory demands. HOSVD presents the worst performance be-

cause it does not run on the clustered tags. That is, it has to process more data, which corresponds to worst time performance. As expected, ClustHOSVD(k-means) outperforms ClustHOSVD(spectral) and ClustHOSVD(hac), because the latter methods require more computations to be made. However, we have to notice that HOSVD(spectral) provides reasonable trade-off between recommendation accuracy and time performance. That is, it insignificantly deteriorates time performance of a factorization, by presenting the best prediction quality. Henceforth, we will use this ClustHOSVD variation to compare with other state-of-the-art tensor-based methods.

6. ACCURACY COMPARISON TO OTHER STATE-OF-THE-ART METHODS

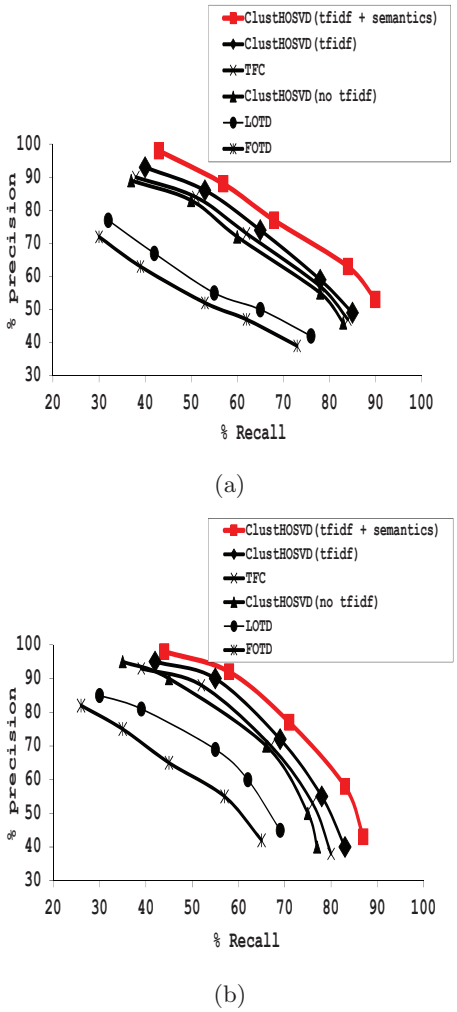
In this Section, we compare our HOSVD(spectral) algorithmic variation, which incorporates the TFIDF weighting schema combined with the semantic similarity of tags denoted henceforth as ClustHOSVD(tfidf + semantics), to the following state-of-the-art methods:

- ClustHOSVD(tfidf) : Our HOSVD(spectral) variation that incorporates only the TFIDF weighting schema.
- TFC : Rafailidis and Daras [22] proposed the TFC model, which is a Tensor Factorization and Tag Clustering model that uses a TFIDF weighting scheme.
- ClustHOSVD(no tfidf) : Leginus et al. [20] proposed tensor-based factorization and tag clustering without applying the TFIDF feature weighting scheme.
- LOTD : Cai et al. [4] proposed Low-Order Tensor Decompositions (LOTD), which is based on low-order polynomial terms on tensors (i.e., first and second order).
- FOTD : Full Order Tensor decomposition proposed by Cai et al. [4] which incorporates except the first and second terms, also the third order polynomial term.

The parameters we used to evaluate the performance of TFC, FOTD, LOTD and ClustHOSVD(no tfidf) are identical to those reported in the original papers. Moreover, Leginus et al. [20] reported spectral k-means clustering method as the one that attains the best accuracy results.

For ClustHOSVD(tfidf + semantics), we vary the weights of a parameter in the similarity measure of Eq. 3, i.e., $a = 1.0$ (considering only semantic similarity) and $a = 0.99, 0.98, ..., 0.01, 0$ (gradual increase of the contribution of cosine similarity which is based on the Vector Space Model). For the Bibsonomy data set, the best performance is achieved by $a = 0.35$, i.e., when both the semantic information and the conventional cosine similarity on tensor unfolding are incorporated into the hybrid similarity function. For the Last.fm data set, the best performance is attained when a parameter is equal to 0.29.

Next, we measure precision versus recall for all six algorithms. The results for the Bibsonomy and Last.fm data sets are depicted in Figures 10a and b, respectively. For both data sets, ClustHOSVD(tfidf + semantics) outperforms the other comparison methods. The reason is that it exploits both conventional cosine similarity and the semantic similarity of tags. In contrast, TFC incorporates the TFIDF weighting scheme without exploiting also semantic information. Moreover, ClustHOSVD(no tfidf) does not apply the TFIDF feature weighting scheme at all. Thus, it does not capture adequately intra-tag similarity and inter-tag dissimilarity. FOTD presents the worst results, which is according to what Cai et al. [4] have reported in their paper. That is, the LOTD method had better results than FOTD in terms of precision-recall diagram, because of the overfitting problem which existed in all data sets.



(a)

(b)

Figure 10: Comparison between variations of ClustHOSVD(tfidf + semantics), ClustHOSVD(tfidf), ClustHOSVD(no tfidf), and LOTD algorithms in terms of precision-recall curve for (a) Bibsonomy and (b) Last.fm data sets.

7. DISCUSSION

In our model, we have used Latent Semantic Indexing (LSI) by performing SVD on the TFIDF weighted W matrix. A probabilistic extension of this idea was introduced by Hofmann [14] as "probabilistic Latent Semantic Indexing" (pLSI). By realizing that pLSI is not a proper generative model at the level of documents, a further extension, Latent Dirichlet Allocation (LDA), was introduced by Blei et al. [3].

An alternative method would be to incorporate those probabilistic techniques in our model instead of TFIDF. However, they are not reported to generally outperform TFIDF and both methods have scalability issues [10]. That is, they work relatively well for small set of training data, which makes them not suitable for our problem, where tag dimension is extremely big (i.e., in the order of million tags).

Someone could argue that HOSVD does not handle missing data, since it treats all (user, item, tag) triplets - that are not seen and are just unknown or missing - as zeros. To address this problem, instead of SVD we can apply kernel-SVD [5] in the three unfolded matrices. Kernel-SVD is the application of SVD in the Kernel-defined feature space and can smooth the severe data sparsity problem.

Notice also that, HOSVD can produce negative values in the reconstructed tensor (while all values in the initial tensor are positive or zero). To avoid this, we could add a constraint of positiveness, as an additional term to the objective function, which will be minimized. For example, the non-negative matrix factorization (NMF) is a group of algorithms in multivariate analysis and linear algebra where a matrix $\mathbf{A}$ is factorized into (usually) two matrices $\mathbf{U}$ and $\mathbf{V}$, with the property that all three matrices have no negative elements. This non-negativity makes the resulting matrices easier to inspect. Since the problem is not exactly solvable in general, it is commonly approximated numerically [2]. Assume that $\mathbf{a}_i, \ldots, \mathbf{a}_N$ are N non-negative input vectors and we organize them as the columns of non-negative data matrix $\mathbf{A}$. Non-negative matrix factorization seeks a small set of K non-negative representative vectors $\mathbf{v}_i, \ldots, \mathbf{v}_K$ that can be non-negatively combine to approximate the input vectors $\mathbf{a}_i$:

$$\mathbf{A} \approx \mathbf{U} \times \mathbf{V} \tag{10}$$

and

$$\mathbf{a}_i \approx \sum_{k=1}^{K} \mathbf{u}_{kn}\mathbf{v}_k, \quad 1 \leq n \leq N\ , \tag{11}$$

where the combining coefficients $\mathbf{u}_{kn}$ and $\mathbf{v}_k$ are restricted to be non-negative[8].

Moreover, someone can claim that HOSVD supports no regularization and thus, it is sensitive to over-fitting. In addition, the appropriate tuning of selected parameters (as described in Section 5.3) can not guarantee the solution of the aforementioned regularization problem. To address this problem, we can extend HOSVD with L2 regularization, which is also known as Tikhonov regularization. After the application of a regularized optimization criterion the possible over-fitting can be reduced. That is, since the basic idea of the HOSVD algorithm is to minimize an element-wise loss on the elements of $\hat{\mathcal{A}}$ by optimizing the square loss, we can extend it with L2 regularization terms.

Finally, someone can argue that when we replace a number of similar tags with a representative tag centroid, we have a loss of granularity and, thus, we may experience lower recommendation accuracy. However, in our case, the "loss" of information from the clustering of similar tags, results in an improvement of data representation, since it handles effectively the problems of synonymy and polysemy. That is, our semantic similarity measure detects synonyms or taxonomically related words, reducing the problem of synonymy; In case of polysemy, we compute the semantic similarity of the possible word meanings, but we cannot ensure that the selected meaning (e.g., the synset that provides the greater similarity value) is the correct one. Thus, the improvement is generally greater in terms of synonymy effects than polysemy. As far as the polysemy is concerned, alternative meanings of a tag could be ignored by our method, which may not be the correct or complete disambiguation of polysemous tags.

8. CONCLUSIONS

This paper extends our previous work on tensor dimensionality reduction [29]. In particular, we combine the clustering of tags with the application of HOSVD on tensors. Our new algorithm pre-groups neighboring tags and produces a set of reduced representative tag centroids, which are inserted in the tensor. The tag pre-grouping is based on two different auxiliary ways to compute the similarity/distance between tags. We compute the cosine similarity of tags based on the TFIDF weighting schema within the vector space model and we also compute their semantic similarity by utilizing the WordNet dictionary. The tag pre-grouping significantly reduces the expense of the tensor decomposition computation, while increasing the accuracy of the item recommendations. Our experiments have shown that spectral clustering outperforms the other clustering methods. In future, we will test our approach on the pre-clustering of items to perform and evaluate the task of tag recommendation. Moreover, we will exploit additional information from WordNet based on the term relations between word senses.

9. REFERENCES

[1] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[2] M. W. Berry, M. Browne, A. N. Langville, V. P. Pauca, and R. J. Plemmons. Algorithms and applications for approximate nonnegative matrix factorization. In *Computational Statistics and Data Analysis*, pages 155–173, 2006.

[3] D. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022, 2003.

[4] Y. Cai, M. Zhang, D. Luo, C. Ding, and S. Chakravarthy. Low-order tensor decompositions for social tagging recommendation. In *Proceedings of the fourth ACM international conference on Web search and data mining (WSDM '11)*, pages 695–704. ACM, 2011.

[5] T.-J. Chin, K. Schindler, and D. Suter. Incremental kernel svd for face recognition with image sets. In *Proceedings of the 7th International Conference on Automatic Face and Gesture Recognition (FGR 2006)*, pages 461–466. IEEE, 2006.

[6] N. Cristianini and J. Shawe-Taylor. Kernel methods for pattern analysis. *Cambridge University Press*, 2004.

[7] I. S. Dhillon and D. S. Modha. Concept decompositions for large sparse text data using clustering. *Machine learning*, 42(1-2):143–175, 2001.

[8] I. S. Dhillon and S. Sra. Generalized nonnegative matrix approximations with bregman divergences. In *In: Neural Information Proc. Systems*, pages 283–290, 2005.

[9] P. Drineas and M. Mahoney. A randomized algorithm for a tensor-based generalization of the singular value decomposition. *Linear algebra and its applications*, 420(2-3):553–571, 2007.

[10] P. Gehler, A. Holub, and M. Welling. The rate adapting poisson model for information retrieval and object recognition. In *Proceedings of the 23rd international conference on Machine learning*, pages 337–344. ACM, 2006.

[11] J. Gemmell, T. Schimoler, B. Mobasher, and R. Burke. Hybrid tag recommendation for social annotation systems. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 829–838. ACM, 2010.

[12] J. Gemmell, A. Shepitsen, B. Mobasher, and R. Burke. Personalization in folksonomies based on tag clustering. *Intelligent techniques for web personalization & recommender systems*, 12, 2008.

[13] J. Herlocker, J. Konstan, L. Terveen, and J. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. on Information Systems*, 22(1):5–53, 2004.

[14] T. Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57. ACM, 1999.

[15] A. Hotho, R. Jaschke, C. Schmitz, and G. Stumme. Information retrieval in folksonomies: Search and ranking. In *The Semantic Web: Research and Applications*, pages 411–426, 2006.

[16] Z. Huang, H. Chen, and D. Zeng. Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering. *ACM Transactions on Information Systems*, 22(1):116–142, 2004.

[17] N. Iakovidou, P. Symeonidis, and Y. Manolopoulos. Multiway spectral clustering link prediction in protein-protein interaction networks. In *Proceedings of the 10th IEEE International Conference on Information Technology and Applications in Biomedicine (ITAB 2010)*, pages 1–4. IEEE, 2010.

[18] T. Kolda and J. Sun. Scalable tensor decompositions for multi-aspect data mining. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM '08)*, pages 363–372, 2008.

[19] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Rev.*, 51(3):455–500, Aug. 2009.

[20] M. Leginus, P. Dolog, and V. Žemaitis. Improving tensor based recommenders with clustering. In *User Modeling, Adaptation, and Personalization Conference (UMAP 2012)*, pages 151–163. Springer, 2012.

[21] J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, page 14. California, USA, 1967.

[22] D. Rafailidis and P. Daras. The tfc model: Tensor factorization and tag clustering for item recommendation in social tagging systems. *Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 2012.

[23] S. Rendle, L. B. Marinho, A. Nanopoulos, and L. S. Thieme. Learning optimal ranking with tensor factorization for tag recommendation. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '09)*, pages 727–736, 2009.

[24] S. Rendle and L. S. Thieme. Pairwise interaction tensor factorization for personalized tag recommendation. In *Proceedings of the third ACM international conference on Web search and data mining (WSDM '10)*, pages 81–90, 2010.

[25] D. Sánchez, M. Batet, D. Isern, and A. Valls. Ontology-based semantic similarity: A new feature-based approach. *Expert Systems with Applications*, 39(9):7718–7728, 2012.

[26] A. Shepitsen, J. Gemmell, B. Mobasher, and R. Burke. Personalized recommendation in social tagging systems using hierarchical clustering. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 259–266. ACM, 2008.

[27] P. Sneath and R. Sokal. *Numerical taxonomy. The principles and practice of numerical classification.* Freeman, 1973.

[28] P. Symeonidis, N. Iakovidou, N. Mantas, and Y. Manolopoulos. From biological to social networks: Link prediction based on multi-way spectral clustering. *Data & Knowledge Engineering*, 87:226–242, 2013.

[29] P. Symeonidis, A. Nanopoulos, and Y. Manolopoulos. A unified framework for providing recommendations in social tagging systems based on ternary semantic analysis. *Knowledge and Data Engineering, IEEE Transactions on*, 22(2):179–192, 2010.

[30] P. Turney. Empirical evaluation of four tensor decomposition algorithms. In *Technical Report (NRC/ERB-1152)*, 2007.

[31] Z. Wu and M. Palmer. Verbs semantics and lexical selection. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 133–138. Association for Computational Linguistics, 1994.

[32] D. Yan, L. Huang, and M. Jordan. Fast approximate spectral clustering. In *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 907–916, 2009.

[33] L. Zelnik-Manor and P. Perona. Self-tuning spectral clustering. In *Advances in neural information processing systems*, pages 1601–1608, 2004.