# Chapter 6
# HOSVD on Tensors and Its Extensions

**Abstract** This chapter describes in detail tensor decomposition for recommender systems. As our running toy example, we will use a tensor with three dimensions (i.e., user–item–tag). The main factorization method that will be presented in this chapter is higher order SVD (HOSVD), which is an extended version of the Singular Value Decomposition (SVD) method. In this chapter, we will present a step-by-step implementation of HOSVD in our toy example. Then we will present how we can update HOSVD when a new user is registered in our recommender system. We will also discuss how HOSVD can be combined with other methods for leveraging the quality of recommendations. Finally, we will study limitations of HOSVD and discuss in detail the problem of non-unique tensor decomposition results and how we can deal with this problem. We also discuss other problems in tensor decomposition, e.g., actualization and scalability.

**Keywords** HOSVD · Higher order singular value decomposition · Tensor decomposition

## 6.1 Algorithm's Outline

In the following, we provide a solid description of the HOSVD method with an outline of the algorithm for the case of social tagging systems, where we have three participatory entities (user, item, and tag). In particular, we provide details of how HOSVD is applied to tensors and how item/tag recommendation is performed based on detected latent associations.

The tensor reduction approach initially constructs a tensor, based on usage data triplets $\{u, i, t\}$ of users, item, and tag. The motivation is to use all the three objects that interact inside a social tagging system. 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 core tensor $\mathcal{S}$ and resulting tensor $\hat{\mathcal{A}}$. The six steps of the HOSVD approach are summarized as follows:

P. Symeonidis and A. Zioupos, *Matrix and Tensor Factorization Techniques for Recommender Systems*, SpringerBriefs in Computer Science,
DOI 10.1007/978-3-319-41357-0_6

- *Step 1*: The initial tensor $\mathcal{A}$ construction, which is based on usage data triplets (user, item, tag).
- *Step 2*: The matrix unfoldings of tensor $\mathcal{A}$, where we matricize the tensor in all three modes, creating three new matrices (one for each mode). (see Eq. 5.1)
- *Step 3*: The application of SVD in all three new matrices, where we keep the $c$-most important singular values for each matrix.
- *Step 4*: The construction of the core tensor $\mathcal{S}$ that reduces dimensionality (see Eq. 5.3).
- *Step 5*: The construction of the $\hat{\mathcal{A}}$ tensor that is an approximation of tensor $\mathcal{A}$ (see Eq. 5.4).
- *Step 6*: Based on the weights of the elements of the reconstructed tensor $\hat{\mathcal{A}}$, we recommend an item/tag to the target user $u$.

Steps 1–5 build a model and can be performed offline. The recommendation in Step 5 is performed online, i.e., each time we have to recommend an item/tag to a user, based on the built model.

## 6.2 HOSVD in STSs

In this section, in order to illustrate how HOSVD works for item recommendation, we apply HOSVD on a toy example. As illustrated in Fig. 6.1, three users tagged three different items (web links). In Fig. 6.1, the part of an arrow line (sequence of arrows with the same annotation) between a user and an item represents that the user tagged the corresponding item, and the part between an item and a tag indicates that the user tagged this item with the corresponding tag. Thus, annotated numbers on arrow lines give the correspondence between the three types of objects. For example, user $u_1$ tagged item $i_1$ with tag "BMW," denoted as $t_1$. The remaining tags are "Jaguar," denoted as $t_2$, "CAT," denoted as $t_3$.

From Fig. 6.1, we can see that users $u_1$ and $u_2$ have common interests on cars, while user $u_3$ is interested in cats. A third-order tensor $\mathcal{A} \in \mathbb{R}^{3\times3\times3}$ can be constructed from usage data. We use the co-occurrence frequency (denoted as weights) of each triplet user, item, and tag as elements of tensor $\mathcal{A}$, which are given in Table 6.1. Note that all associated weights are initialized to 1. Figure 6.2 shows the tensor construction of our running example.

**Table 6.1** Associations of the running example

| Arrow line | User | Item | Tag | Weight |
|---|---|---|---|---|
| 1 | $u_1$ | $i_1$ | $t_1$ | 1 |
| 2 | $u_2$ | $i_1$ | $t_1$ | 1 |
| 3 | $u_2$ | $i_2$ | $t_2$ | 1 |
| 4 | $u_3$ | $i_3$ | $t_3$ | 1 |

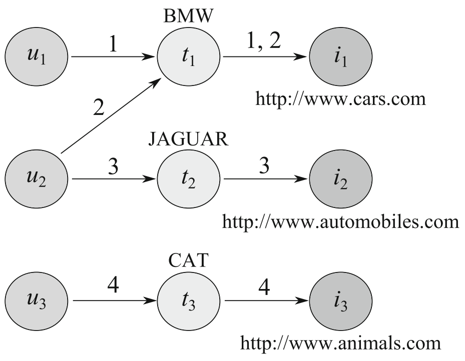


**Fig. 6.1** Usage data of the running example

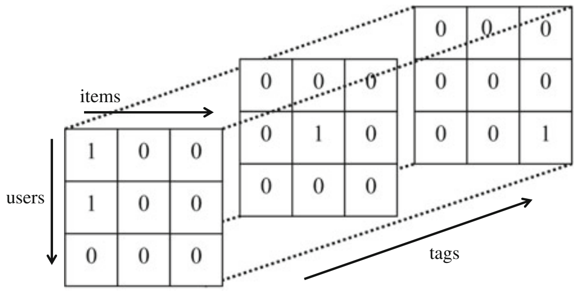


**Fig. 6.2** The tensor construction of our running example

After performing tensor reduction analysis, we can get the reconstructed tensor of $\hat{\mathcal{A}}$, which is presented in Table 6.2, whereas Fig. 6.3 depicts the contents of $\hat{\mathcal{A}}$ graphically (weights are omitted). As shown in Table 6.2 and Fig. 6.3, the output of the tensor reduction algorithm for the running example is interesting, because a new association among these objects is revealed. The new association is between $u_1$, $i_2$, and $t_2$. It is represented with the last (boldfaced) row in Table 6.2 and with the dashed arrow line in Fig. 6.3.

If we have to recommend to $u_1$ an item for tag $t_2$, then there is no direct indication for this task in the original tensor $\mathcal{A}$. However, we see that in Table 6.2 the element of $\hat{\mathcal{A}}$ associated with $(u_1, i_2, r_2)$ is 0.44, whereas for $u_1$, there is no other element associating other tags with $i_2$. Thus, we recommend item $i_2$ to user $u_1$, who used tag $t_2$. For the current example, the resulting $\hat{\mathcal{A}}$ tensor is shown in Fig. 6.4.

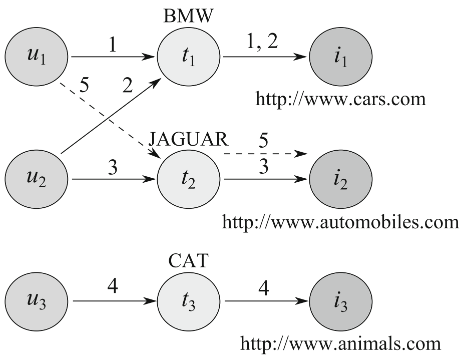


**Fig. 6.3** Illustration of the tensor reduction algorithm output for the running example

**Table 6.2** Associatings derived on the running example

| Arrow line | User | Item | Tag | Weight |
|---|---|---|---|---|
| 1 | $u_1$ | $i_1$ | $t_1$ | 0.72 |
| 2 | $u_2$ | $i_1$ | $t_1$ | 1.17 |
| 3 | $u_2$ | $i_2$ | $t_2$ | 0.72 |
| 4 | $u_3$ | $i_3$ | $t_3$ | 1 |
| **5** | $\mathbf{u_1}$ | $\mathbf{i_2}$ | $\mathbf{t_2}$ | **0.44** |

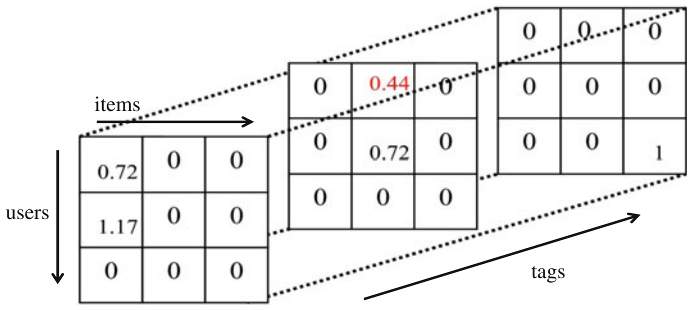


**Fig. 6.4** The resulting $\hat{\mathcal{A}}$ tensor for the running example

The resulting recommendation is reasonable, because $u_1$ is interested in cars rather than cats. That is, the tensor reduction approach is able to capture latent associations among the multitype data objects: user, items, and tags. The associations can then be used to improve the item recommendation procedure.

### 6.2.1 Handling the Sparsity Problem

Sparsity is a severe problem in three-dimensional data, which could affect the outcome of SVD. To address this problem, instead of SVD we can apply kernel SVD [2, 3] in three unfolded matrices. Kernel SVD is the application of SVD in the kernel-defined feature space. Smoothing with kernel SVD is also applied by Symeonidis et al. in [11].

For each unfolding $A_i$ $(1 \leq i \leq 3)$, we have to nonlinearly map its contents to a higher dimensional space using a mapping function $\phi$. Therefore, from each $A_i$ matrix we derive an $F_i$ matrix, where each element $a_{xy}$ of $A_i$ is mapped to the corresponding element $f_{xy}$ of $F_i$, i.e., $f_{xy} = \phi(a_{xy})$. Next, we can apply SVD and decompose each $F_i$ as follows:

$$F_i = U^{(i)} S^{(i)} (V^{(i)})^T \tag{6.1}$$

The resulting $U^{(i)}$ matrices are then used to construct the core tensor.

Nevertheless, to avoid explicit computation of $F_i$, all computations must be done in the form of inner products. In particular, as we are interested to compute only matrices with left singular vectors, for each mode $i$ we can define a matrix $B_i$ as follows:

$$B_i = F_i F_i^T \tag{6.2}$$

As $B_i$ is computed using inner products from $F_i$, we can substitute the computation of inner products with the results of a kernel function. This technique is called the "kernel trick" [3] and avoids the explicit (and expensive) computation of $F_i$. As each $U^{(i)}$ and $V^{(i)}$ are orthogonal and each $S^{(i)}$ is diagonal, it easily follows from Eqs. 6.1 and 6.2 that:

$$B_i = (U^{(i)} S^{(i)} (V^{(i)})^T)(U^{(i)} S^{(i)} (V^{(i)})^T)^T = U^{(i)} (S^{(i)})^2 (V^{(i)})^T \tag{6.3}$$

Therefore, each required $U^{(i)}$ matrix can be computed by diagonalizing each $B_i$ matrix (which is square) and taking its eigenvectors.

Regarding the kernel function, in our experiments, we use the Gaussian kernel $K(x, y) = e^{-\frac{||x-y||^2}{c}}$, which is commonly used in many applications of kernel SVD. As Gaussian kernel parameter $c$, we use the estimate for standard deviation in each matrix unfolding.

### 6.2.2 Inserting New Users, Tags, or Items

As new users, tags, or items are being introduced to the system, the tensor $\hat{\mathcal{A}}$, which provides the recommendations, has to be updated. The most demanding operation is the updating of SVD of the corresponding mode in Eqs. 6.1 and 6.3. As we would like to avoid the costly batch recomputation of the corresponding SVD, we can consider incremental solutions [1, 9]. Depending on the size of the update (i.e., number of new users, tags, or items), different techniques have been followed in related research. For small update sizes, we can consider the *folding-in* technique [4, 9], whereas for larger update sizes, we can consider incremental SVD techniques [1]. Both techniques are described next [11].

### *6.2.3 Update by Folding-in*

Given a new user, we first compute the new 1-mode matrix unfolding $A_1$. It is easy to see that entries of the new user result in appending of a new row in $A_1$. This is exemplified in Fig. 6.5. Figure 6.5a shows the insertion of a new user in the tensor of the current example (new values are presented with red color). Notice that to ease presentation, new user tags and items are identical to those of user $U_2$.

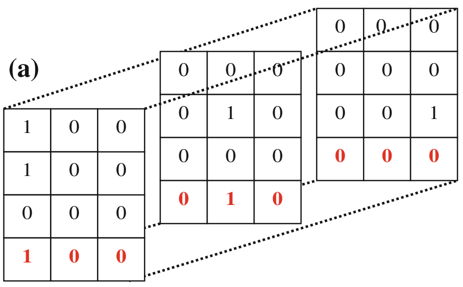
Let $\mathbf{u}$ denote the new row that is appended to $A_1$. Figure 6.5b shows the new $A_1$, i.e., the 1-mode unfolded matrix, where it is shown that contents of $\mathbf{u}$ (highlighted with red color) have been appended as a new row in the end of $A_1$.

Since $A_1$ changed, we have to compute its SVD, as given in Eq. 6.5. To avoid a batch SVD recomputation, we can use the existing basis $U_{c1}^{(1)}$ of left singular vectors to project the $\mathbf{u}$ row onto the reduced $c1$-dimensional space of users in the $A_1$ matrix. This projection is called folding-in and is computed using the following Eq. 6.4 [4]:

$$\mathbf{u_{new}} = \mathbf{u} \cdot V_{c1}^{(1)} \cdot (S_{c1}^{(1)})^{-1} \tag{6.4}$$

In Eq. 6.4, $\mathbf{u_{new}}$ denotes the mapped row, which will be appended to $U_{c1}^{(1)}$, whereas $V_{c1}^{(1)}$ and $(S_{c1}^{(1)})^{-1}$ are dimensionally reduced matrices derived when SVD was originally applied to $A_1$, i.e., before insertion of the new user. In the current example, computation of $\mathbf{u_{new}}$ is described in Fig. 3.4.

The $\mathbf{u_{new}}$ vector should be appended to the end of the $U_{c1}^{(1)}$ matrix. For the current example, appending should be done to the previously $U_{c1}^{(1)}$ matrix. Notice that in the example, $\mathbf{u_{new}}$ is identical with the second column of the transpose of $U_{c1}^{(1)}$. The reason is that the new user has identical tags and items with user $U_2$ and we mapped them on the same space (recall that the folding-in technique maintains the same space computed originally by SVD) (Fig. 6.6).



**(b)**

$$A_1 = \begin{bmatrix} 1\,0\,0\,0\,0\,0\,0\,0\,0 \\ 1\,0\,0\,0\,1\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,0\,0\,1 \\ \mathbf{1\,0\,0\,0\,1\,0\,0\,0\,0} \end{bmatrix}$$

**Fig. 6.5** Example of folding in a new user: **a** the insertion of a new user in tensor, **b** the new 1-mode unfolded matrix $\mathbf{A}_1$

$$\underbrace{\begin{bmatrix} -0.85 & 0 \end{bmatrix}}_{\mathbf{u_{new}}} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{u}} \times$$

$$\times \underbrace{\begin{bmatrix} -0.85 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -0.53 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}}_{\mathbf{V}_{c1}^{(1)}} \times \underbrace{\begin{bmatrix} 0.62 & 0 \\ 0 & 1 \end{bmatrix}}_{(\mathbf{S}_{c1}^{(1)})^{-1}}$$

**Fig. 6.6** The result of folding-in for the current example

Finally, to update the tensor $\hat{\mathcal{A}}$, we have to perform products given in Eq. 6.1. Notice that only $U_{c_1}^{(1)}$ has been modified in this equation. Thus, to optimize insertion of new users, as mode products are interchangeable, we can perform this product as $\left[\mathcal{S} \times_2 U_{c_2}^{(2)} \times_3 U_{c_3}^{(3)}\right] \times_1 U_{c_1}^{(1)}$, where the left factor (inside the brackets), which is unchanged, can be prestored so as to avoid its recomputation. For the current example, the resulting $\hat{\mathcal{A}}$ tensor is shown in Fig. 6.7.

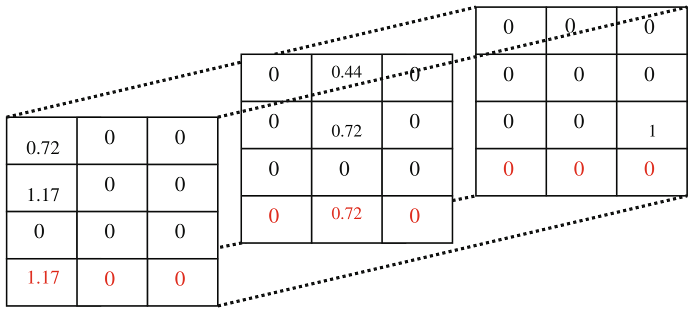


**Fig. 6.7** The resulting $\hat{\mathcal{A}}$ tensor of running example after the insertion of new user

An analogous insertion procedure can be followed for the insertion of a new item or tag. For a new item insertion, we have to apply Eq. 6.4 on the 2-mode matrix unfolding of tensor $\mathcal{A}$, while for a new tag, we apply Eq. 6.4 on the 3-mode matrix unfolding of tensor $\mathcal{A}$.

### *6.2.4 Update by Incremental SVD*

Folding-in incrementally updates SVD, but the resulting model is not a perfect SVD model, because the space is not orthogonal [9]. When the update size is not big, the loss of orthogonality may not be a severe problem in practice. Nevertheless, for larger update sizes the loss of orthogonality may result in an inaccurate SVD model. In this case, we need to incrementally update SVD so as to ensure orthogonality. This can be attained in several ways. Next, we describe the approach proposed by Brand [1].

Let $M_{p\times q}$ be a matrix, upon we which apply SVD and maintain the first $r$ singular values, i.e.,

$$M_{p\times q} = U_{p\times r} S_{r\times r} V^T_{r\times q} \tag{6.5}$$

Assume that each column of matrix $C_{p\times c}$ contains additional elements. Let $L = U\backslash C = U^T C$ be the projection of $C$ onto the orthogonal basis of $U$. Let also $H = (I - UU^T)C = C - UL$ be the component of $C$ orthogonal to the subspace spanned by $U$ ($I$ is the identity matrix). Finally, let $J$ be an orthogonal basis of $H$ and let $K = J\backslash H = J^T H$ be the projection of $C$ onto subspace orthogonal to $U$. Consider the following identity:

$$[U\ J]\begin{bmatrix} S & L \\ 0 & K \end{bmatrix}\begin{bmatrix} V & 0 \\ 0 & I \end{bmatrix}^T = \begin{bmatrix} U(I - UU^T)C/K \end{bmatrix}\begin{bmatrix} S & U^T C \\ 0 & K \end{bmatrix}\begin{bmatrix} V & 0 \\ 0 & I \end{bmatrix}^T = \begin{bmatrix} USV^T & C \end{bmatrix} = [M\ C]$$

Like an SVD, left and right matrices in the product are unitary and orthogonal. The middle matrix, denoted as $Q$, is diagonal. To incrementally update SVD, $Q$ must be diagonalized. If we apply SVD on $Q$, we get:

$$Q = U'S'(V')^T \tag{6.6}$$

Additionally, define $U'', S'', and V''$ as follows:

$$U'' = [U \;\; J]U', \;\; S'' = S', \;\; V'' = \begin{bmatrix} V & 0 \\ 0 & I \end{bmatrix} V' \tag{6.7}$$

Then, the updated SVD of matrix $[M \;\; C]$ is as follows:

$$[M \;\; C] = [USV^T \;\; C] = U''S''(V'')^T \tag{6.8}$$

This incremental update procedure takes $O((p+q)r^2 + pc^2)$ time.

Returning to the application of an incremental update for new users, items, or tags, as described in Sect. 6.2.3, in each case it resulted in a number of new rows that are appended to the end of the unfolded matrix of the corresponding mode. Therefore, we need an incremental SVD procedure in the case where we add new rows, whereas the aforementioned method works in the case where we add new columns. In this case, we simply swap $U$ for $V$ and $U''$ for $V''$.

## 6.3 Limitations and Extensions of HOSVD

In this section, we discuss some limitations of HOSVD and describe other extensions of the HOSVD method. In particular, we discuss in detail the problem of non-unique tensor decomposition results and how we can deal with this problem. We also discuss other problems in tensor decomposition, e.g., missing data, scalability, and overfitting.

As far as scalability is concerned, the runtime complexity is cubic in the size of latent dimensions. This is shown in Eq. 5.6, where three nested sums have to be calculated just for predicting a single (user, item, tag) triplet. In the direction of solving this scalability issue, there are approaches to improve the efficiency of HOSVD [5, 12].

As far as non-unique of tensor decompositions (HOSVD and PARAFAC) results is concerned, since their objective functions are nonconvex, there are a large number of local optima. That is, starting from different starting points, the iteratively improved solution may converge to different local solutions (see Sect. 5.3). Duo et al. [6] have experimentally shown that for all real-life data sets they tested, the HOSVD solution is unique (i.e., different initial starting points always converge to a unique global solution), whereas the PARAFAC solution is almost always not unique.

Since HOSVD solutions are unique, it means that the resulting approximation tensor is repeatable and reliable.

Someone could argue that HOSVD does not handle missing data, since it treats all (user, item, and 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 [2] 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. In the same direction, lexical similarity between tags can further downsize the sparsity problem. That is, by considering also the synonymity of tags, we can increase the number of nonzero elements in the tensor.

Finally, someone could claim that HOSVD supports no regularization, and thus, it is sensitive to overfitting. In addition, appropriate tuning of selected parameters cannot guarantee a solution to 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, possible overfitting can be reduced. That is, since the basic idea of the HOSVD algorithm is to minimize an elementwise loss on elements of $\hat{\mathcal{A}}$ by optimizing the square loss, we can extend it with L2 regularization terms.

## 6.3.1 Combining HOSVD with a Content-Based Method

Social tagging has become increasingly popular in music information retrieval (MIR). It allows users to tag music resources such as songs, albums, or artists. Social tags are valuable to MIR, because they comprise a multifaceted source of information about genre, style, mood, users' opinion, or instrumentation.

Symeonidis et al. [8] examined the problem of personalized song recommendation (i.e., resource recommendation) based on social tags. They proposed the modeling of social tagging data with three-order tensors, which capture cubic (three-way) correlations between users–tags–music items. The discovery of a latent structure in this model is performed with HOSVD, which helps to provide accurate and personalized recommendations, i.e., adapted to particular users' preferences.

However, the aforementioned model suffers from sparsity that incurs in social tagging data. Thus, to further improve the quality of recommendation, Nanopoulos et al. [7] enhanced the HOSVD model with a tag-propagation scheme that uses similarity values computed between music resources based on audio features. As a result, this hybrid model effectively combines both information about social tags and audio features. Nanopoulos et al. [7] examined experimentally the performance of the proposed method with real data from Last.fm. Their results indicate superiority of the proposed approach compared to existing methods that suppress cubic relationships that are inherent in social tagging data. Additionally, their results suggest that combination of social tagging data with audio features is preferable to use the former alone.

### *6.3.2 Combining HOSVD with a Clustering Method*

In this section, we describe how we can combine HOSVD with the clustering of tags in STSs. In this direction, Panagiotis Symeondis [10] proposed an effective preprocessing step, i.e., the clustering of tags, that can reduce the size of tensor dimensions and deal with its missing values. To perform clustering of tags, he initially incorporates in his model two different auxiliary ways to compute similarity/distance between tags. First, he computes cosine similarity of tags based on term frequency-inverse document frequency within the vector space model. Second, to address polysemy and synonymity of tags, he also computes their semantic similarity by utilizing the WordNet[1] dictionary.

After clustering tags, he uses centroids of found tag clusters as representatives for tensor tag dimension. As a result, he efficiently overcame the tensor's computational bottleneck by reducing both factorization dimension and data sparsity. Moreover, clustering of tags is an effective means to reduce tag ambiguity and tag redundancy, resulting in better accuracy prediction and item recommendations. He used three different clustering methods (i.e., k-means, spectral clustering, and hierarchical agglomerative clustering) for discovering tag clusters.

The main intuition of combining HOSVD with a clustering method (e.g., spectral clustering, k-means, etc.) in STSs is based on the fact that if we perform tag clustering before tensor construction, we will be able to build a lower dimension tensor based on found tag clusters. The ClustHOSVD algorithm consists of three main parts: (i) tag clustering, (ii) tensor construction and its dimensionality reduction, and (iii) item recommendation based on detected latent associations. Figure 6.8 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 in. The output is the reduced approximate tensor which incorporates the tag cluster dimension and a set of recommended items to user $u$.

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

```
Algorithm 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 decompose and recompose the $\mathcal{A}$ tensor.
        (Steps 1–6 of Section 6.1).
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 $t$.
```

**Fig. 6.8** Outline of the ClustHOSVD Algorithm

In step 1, 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 an $f$-dimensional vector. In case we apply multiway spectral clustering, we can apply first $k$-means to cluster the tags of the tripartite graph, and then we can apply spectral clustering only on cluster centroids (representative tags of each cluster). Using this implementation, the overall computation cost of multiway 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, runtime complexity of HOSVD is cubic in the size of latent dimensions. However, ClustHOSVD algorithm performs clustering on the tag dimension, resulting usually in a small number of tag clusters. Notice that the same procedure can be followed for other two dimensions (users and items). Thus, it can result a tensor with a very small number of latent dimensions.

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

## References

1. Burke, R.: Hybrid recommender systems: survey and experiments. User Model. User-Adap. Inter. **12**(4), 331–370 (2002)
2. Chin, T.-J., Schindler, K., Suter, D.: Incremental kernel svd for face recognition with image sets. In: Proceedings of the 7th International Conference on Automatic Face and Gesture Recognition (FGR 2006), pp. 461–466. IEEE (2006)
3. Cristianini, N., Shawe-Taylor, J.: Kernel Methods for Pattern Analysis. Cambridge University Press (2004)
4. Furnas, G., Deerwester, S., Dumais, S., et al.: Information retrieval using a singular value decomposition model of latent semantic structure. In: Proceedings of ACM SIGIR Conference, pp. 465–480 (1988)
5. Kolda, T.G., Sun, J.: Scalable tensor decompositions for multi-aspect data mining. In: ICDM'08: Proceedings of the 8th IEEE International Conference on Data Mining, pp. 363–372. IEEE Computer Society, Dec 2008
6. Luo, D., Ding, C., Huang, H.: Are tensor decomposition solutions unique? on the global convergence hosvd and parafac algorithms. In: Advances in Knowledge Discovery and Data Mining, pp. 148–159. Springer (2011)
7. Nanopoulos, A., Rafailidis, D., Symeonidis, P., Manolopoulos, Y.: Musicbox: personalized music recommendation based on cubic analysis of social tags. IEEE Trans. Audio Speech Lang. Process. **18**(2), 407–412 (2010)
8. Nanopoulos, A., Symeonidis, P., Ruxanda, M., Manolopoulos, Y.: Ternary semantic analysis of social tags for personalized music recommendation. In: ISMIR'08: Proceedings of the 9th ISMIR Conference, New York, pp. 219–224 (2008)
9. Sarwar, B., Konstan, J., Riedl, J.: Incremental singular value decomposition algorithms for highly scalable recommender systems. In: International Conference on Computer and Information Science (2002)
10. Symeonidis, P.: ClustHOSVD: item recommendation by combining semantically enhanced tag clustering with tensor HOSVD. IEEE Syst. Man Cybern. (2015)
11. Symeonidis, P., Nanopoulos, A., Manolopoulos, Y.: A unified framework for providing recommendations in social tagging systems based on ternary semantic analysis. IEEE Trans. Knowl. Data Eng. **22**(2) (2010)
12. Turney, P.: Empirical evaluation of four tensor decomposition algorithms. Technical report (NRC/ERB-1152) (2007)