Yours Truly

Graph-Based Social Media Analysis

Contents

1 Matrix and Tensor Factorization with Applications **1**

1.1 Introduction 1
1.2 Singular Value Decomposition on Matrices for recommender systems 3
1.2.1 Applying the SVD and Preserving the Largest Singular Values 4
1.2.2 Generating the Neighborhood of Users/Items 5
1.2.3 Generating the Recommendation List 6
1.2.4 Inserting a Test User in the c-dimensional Space . . . 6
1.2.5 Other Factorization Methods 7
1.3 Higher Order Singular Value Decomposition (HOSVD) on Tensors 8
1.3.1 From SVD to HOSVD 9
1.3.2 HOSVD for Recommendations in Social Tagging Systems (STSs) 12
1.3.3 Handling the Sparsity Problem 16
1.3.4 Inserting new users, tags, or items 17
1.3.4.1 Update by folding-in 17
1.3.4.2 Update by Incremental SVD 20
1.3.5 Other Scalable Factorization Models 21
1.4 A Real Geo-Social System based on HOSVD 22
1.4.1 GeoSocialRec Web Site 23
1.4.2 GeoSocialRec Database and Recommendation Engine 25
1.4.3 Experiments 25
1.5 Conclusion 28

Bibliography **29**

Index **33**

Chapter 1

Matrix and Tensor Factorization with Applications

Abstract 1
1.1 Introduction 1
1.2 Singular Value Decomposition on Matrices for recommender systems 2
1.2.1 Applying the SVD and Preserving the Largest Singular Values 4
1.2.2 Generating the Neighborhood of Users/Items 5
1.2.3 Generating the Recommendation List 6
1.2.4 Inserting a Test User in the c-dimensional Space 6
1.2.5 Other Factorization Methods 7
1.3 Higher Order Singular Value Decomposition (HOSVD) on Tensors 8
1.3.1 From SVD to HOSVD 9
1.3.2 HOSVD for Recommendations in Social Tagging Systems (STSs) 12
1.3.3 Handling the Sparsity Problem 16
1.3.4 Inserting new users, tags, or items 17
1.3.4.1 Update by folding-in 17
1.3.4.2 Update by Incremental SVD 20
1.3.5 Other Scalable Factorization Models 21
1.4 A Real Geo-Social System based on HOSVD 22
1.4.1 GeoSocialRec Web Site 23
1.4.2 GeoSocialRec Database and Recommendation Engine . 25
1.4.3 Experiments 25
1.5 Conclusion 28

Abstract

Representing data in lower dimensional spaces has been extensively used in many disciplines such as natural language and image processing, data mining and information retrieval [DDF+90]. Recommender systems deal with challenging issues such as scalability, noise, and sparsity and thus, matrix and

tensor factorization techniques appear as an interesting tool to be exploited. Symeonidis et al. [SNPM06, Sym07], for example, used SVD for the prediction of items/ratings in recommender systems. They assumed that there is only a small number of factors influencing the users' preferences, and that a user's preference for an item is determined by how each factor applies to the user and the item. More recently, due to the Netflix challenge[1], research on matrix factorization methods, a class of latent factor models, gained renewed momentum in the recommender systems literature, given that many of the best performing methods on the challenge were based on matrix factorization techniques [Kor08, SM08, Kor09]. In this chapter we describe matrix and tensor factorization techniques (i.e. SVD on matrices and HOSVD on tensors) in recommender systems and social tagging systems, respectively. In addition, we present a real-world recommender system for Location-Based Social Networks, which employs tensor decomposition techniques.

1.1 Introduction

While technology is developed fast, data become larger as well, and as the size of data grows so does the difficulty of processing it. One way to view and process data easily is as a matrix, which is a rectangular array of numbers. However, matrices suffer from big data as well. The dimensions of matrices keep on growing fast and this fact make analysts' job more difficult. The problem of dimensionality reduction appears when data are in fact of a higher dimension than being manageable. Dimensionality reduction attempts to reduce the dimensionality of data to manageable size, while keeping as much of the original important information as possible.

The "information overload" problem affects our everyday experience while searching for knowledge on a topic. To overcome this problem, we often rely on suggestions from others who have more experience on the topic. In Web, this is attained with the usage of Collaborative Filtering (CF), which provides recommendations based on the suggestions of users who have similar preferences. Since CF is able to capture the particular preferences of a user, it has become one of the most popular methods in recommender systems. The classic CF (i.e., user-based CF and item-based CF) methods are also known as memory-based methods and constitute the first subgroup of the categorization of CF systems. Memory-based methods firstly load into the main memory the rating matrix and afterwards provide recommendations based on the relationship between user-item pair and the rest of the rating matrix.

[1]The Netflix challenge was a competition for the best recommender system algorithm to predict user ratings for movies. The competition was held by Netflix (http://www.netflixprize.com/), an on-line DVD-rental service.

The second main category of CF algorithms is known as model-based algorithms, which recommend by first developing a model of user ratings for items. These methods fit a parameterized model to the given rating matrix and provide recommendations based on the fitted model. It has been shown that, model-based algorithms can efficiently handle scalability and improve accuracy of recommendations in large data sets. Model-based approaches can combine the effectiveness of the nearest-neighbor CF algorithms in terms of accuracy, with the efficiency in terms of execution time.

Towards this direction, SVD is a technique that has been extensively used in informational retrieval. It detects latent relationships between documents and terms. In CF, SVD can be used to form users' trends from individual preferences, by detecting latent relationships between users and items. Therefore, with SVD, a higher level representation of the original user-item matrix is produced, which presents a three-fold advantage: (i) it contains the main trends of users' preferences, (ii) noise is removed, (iii) it is much more condensed than the original matrix, thus it favors scalability.

In the following, we describe matrix and tensor factorization techniques in Sections 1.2 and 1.3, respectively. Finally, in Section 1.4, we present a real-world recommender system, which is based on HOSVD.

1.2 Singular Value Decomposition on Matrices for recommender systems

A well-known latent factor model for matrix decomposition is singular value decomposition. The singular value decomposition (SVD) [Str06] of a matrix $A_{I_1 \times I_2}$ can be written as a product of three matrices, as shown in Equation 1.1:

$$A_{I_1 \times I_2} = U_{I_1 \times I_1} \cdot S_{I_1 \times I_2} \cdot V^{\top}_{I_2 \times I_2}, \tag{1.1}$$

where U is the matrix with the left singular vectors of A, V' is the transpose of the matrix V with the right singular vectors of A, and S is the diagonal matrix of ordered singular values [2] of A.

By preserving only the largest $c < \min\{I_1, I_2\}$ singular values of S, SVD results in matrix $\hat{A}$, which is an approximation of A. In information retrieval, this technique is used by LSI [FDD+88], to deal with the latent semantic associations of terms in texts and to reveal the major trends in A.

To perform the SVD over a user-item matrix A, we tune the value of parameter c, of singular values (i.e., dimensions) with the objective to reveal the major trends. The tuning of c is determined by the rank of matrix A. A rule of thumb, for defining parameter c is to compute the sum of elements in

[2] The singular values determined by the factorization of Equation 1.1 are unique and satisfy $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \cdots \geq \sigma_{I_2} \geq 0$.

the main diagonal of S matrix (also known as the nuclear norm). Next, we preserve a sufficient percentage of this sum for the creation of an approximation of the original matrix A. If we have the allowance to use less information percentage with the similar results, we just have to reduce the value of c and sum the corresponding elements of the main diagonal of S matrix. Therefore, a c-dimensional space is created and each of the c dimensions corresponds to a distinctive rating trend. Next, given the current ratings of the target user u, we enter pseudo-user vector in the c-dimensional space. Finally, we find the k nearest neighbors of pseudo user vector in the c-dimensional space and apply either user- or item-based similarity to compute the top-N recommended items. Conclusively, the provided recommendations consider the existence of user rating trends, as the similarities are computed in the reduced c-dimensional space, where dimensions correspond to trends.

To ease the discussion, we will use the running example illustrated in Figure 1.2.1 where I_{1-4} are items and U_{1-4} are users. As shown, the example data set is divided into training and test set. The null cells(no rating) are presented as zeros.

	I_1	I_2	I_3	I_4
U_1	4	1	1	4
U_2	1	4	2	0
U_3	2	1	4	5

(a)

	I_1	I_2	I_3	I_4
U_4	1	4	1	0

(b)

FIGURE 1.2.1: (a) Training Set (3×4), (b) Test Set (1×4).

1.2.1 Applying the SVD and Preserving the Largest Singular Values

Initially, we apply the SVD to a $n \times m$ matrix A (i.e., the training data of our running example) that produces the decomposition shown in Equation 1.2. The matrices of our running example are shown in Figure 1.2.2.

$$A_{n\times m} = U_{n\times n} \cdot S_{n\times m} \cdot V^{\top}_{m\times m} \tag{1.2}$$

It is possible to reduce the $n \times m$ matrix S to have only c largest singular values. Then, the reconstructed matrix is the closest rank-c approximation of the initial matrix A, as it is shown in Equation 1.3 and Figure 1.2.3:

$$A^{*}_{n\times m} = U_{n\times c} \cdot S_{c\times c} \cdot V^{\top}_{c\times m} \tag{1.3}$$

We tune the number, c, of singular values (i.e., dimensions) with the objective to reveal the major trends. The tuning of c is determined by the information percentage that is preserved compared to the original matrix. Therefore, a c-dimensional space is created and each of the c dimensions corresponds

4	1	1	4
1	4	2	0
2	1	4	5

$\mathbf{A}_{n\times m}$

-0.61	0.28	-0.74
-0.29	-0.95	-0.12
-0.74	0.14	0.66

$\mathbf{U}_{n\times n}$

8.87	0	0	0
0	4.01	0	0
0	0	2.51	0

$\mathbf{S}_{n\times m}$

-0.47	-0.28	-0.47	-0.69
0.11	-0.85	-0.27	0.45
-0.71	-0.23	0.66	0.13
-0.52	0.39	-0.53	0.55

$\mathbf{V'}_{m\times m}$

FIGURE 1.2.2: Example of: $\mathbf{A}_{n\times m}$ (initial matrix $\mathbf{A}$), $\mathbf{U}_{n\times m}$ (left singular vectors of $\mathbf{A}$), $\mathbf{S}_{n\times m}$ (singular values of $\mathbf{A}$), $\mathbf{V'}_{m\times m}$ (right singular vectors of $\mathbf{A}$).

2.69	0.57	2.22	4.25
0.78	3.93	2.21	0.04
3.17	1.38	2.92	4.78

$\mathbf{A^*}_{n\times i}$

-0.61	0.28
-0.29	-0.95
-0.74	0.14

$\mathbf{U}_{n\times c}$

8.87	0
0	4.01

$\mathbf{S}_{c\times c}$

-0.47	-0.28	-0.47	-0.69
0.11	-0.85	-0.27	0.45

$\mathbf{V'}_{c\times m}$

FIGURE 1.2.3: Example of: $\mathbf{A^*}_{n\times m}$ (approximation matrix of $\mathbf{A}$), $\mathbf{U}_{n\times c}$ (left singular vectors of $\mathbf{A}^*$), $\mathbf{S}_{c\times c}$ (singular values of $\mathbf{A}^*$), $\mathbf{V'}_{c\times m}$ (right singular vectors of $\mathbf{A}^*$).

to a distinctive rating trend. We have to notice that in the running example we create a 2-dimensional space using 83,7% of the total information of the matrix (12,88/15,39) [3].

1.2.2 Generating the Neighborhood of Users/Items

Having a reduced dimensional representation of the original space [4], we form the neighborhoods of users/items in that space. In particular, there are

[3] 15,39 is the sum of elements in the main diagonal of $S_{c\times c}$ (singular values of A^*)

[4] The original space consists of two subspaces:
- range of (A) whose U (see Fig. 1.2.3) is an orthonormal basis. This vector space is the column space of A referred to users.

two subspaces: The first is the range of A, whose matrix $U_{n\times c}$ is its orthonormal basis. This vector space is the column space of A and refers to users. The second is the range of A^T, whose matrix $V_{m\times c}$ is its orthonormal basis. This vector space is the row space of A and refers to items.

A user-based approach relies on the predicted value of a rating that a user gives on an item I_j. This value is computed as an aggregation of the ratings of the user's neighborhood (i.e., similar users) on this particular item. Whereas, an item-based approach takes under consideration only the user-item rating matrix (e.g., a user rated a movie with a rate of 3).

For the user based approach, we find the k nearest neighbors of pseudo user vector in the c-dimensional space. The similarities between training and test users can be based on Cosine Similarity. First, we compute the matrix $U_{n\times c} \cdot S_{c\times c}$ and then we perform vector similarity among rows. This $n \times c$ matrix is the c-dimensional representation for the n users.

For the item based approach, we find the k nearest neighbors of item vector in the c-dimensional space. First, we compute the matrix $S_{c\times c} \cdot V^{\top}_{c\times m}$ and then we perform vector similarity among columns. This $c \times m$ matrix is the c-dimensional representation for the m items.

1.2.3 Generating the Recommendation List

The most often used technique for the generation of the top-N list of recommended items, is the one that counts the frequency of each item inside the found neighborhood, and recommends the N most frequent ones. Henceforth, this technique is denoted as Most-Frequent item recommendation (MF). Based on MF, we sort (in descending order) the items according to their frequency in the found neighbourhood of the target user, and recommend the first N of them.

As another method, someone could use the predicted values for each item to rank them. This ranking criterion, denoted as Highest Predicted Rated item recommendation (HPR), is influenced by the the Mean Absolute Error (MAE) between the predicted and the real preferences of a user for an item. [5]. HPR opts for recommending the items that are more probable to receive a higher rating. Notice that HPR presents poor performance for the classic CF algorithms. However, it has very good results when it is used in combination with SVD. The reason is that in the latter it is based only on the major trends of users.

As another method, we can sum the positive rates of the items in the neighborhood, instead of just counting their frequency. This method is denoted as highest sum of rates item recommendation (HSR). The top-N list consists of the N items with the highest sum. The intuition behind HSR is that it takes

- range of $(A^{\top})$ whose V (see Fig. 1.2.3) is an orthonormal basis. This vector space is the row space of A referred to items.

[5] $MAE = \frac{1}{n}\sum_{i=1}^{n}|f_i - y_i|$: The Mean Absolute Error (MAE) is the average of the absolute errors $e_i = f_i - y_i$, where f_i is the prediction and y_i the true value.

into account both the frequency (as MF) and the actual ratings, because it wants to favor items that appear most frequently in the neighborhood and have the best ratings. Assume, for example, an item I_j that has just a smaller frequency than an item I_k. If I_j is rated much higher than I_k, then HSR will prefer it from I_k, whereas MF will favor I_k.

1.2.4 Inserting a Test User in the c-dimensional Space

Related work [SKKR00] has studied SVD on CF considering the test data as apriori known. It is evident that, for user-based approach, the test data should be considered as unknown in the c-dimensional space. Thus a specialized insertion process should be used. Given the current ratings of the test user u, we enter pseudo-user vector in the c-dimensional space using the following Equation 1.4 [FDD+88]. In the current example,we insert U_4 into the 2-dimensional space, as it is shown in Figure 1.2.4:

$$\mathbf{u_{new}} = \mathbf{u} \cdot V_{m \times c} \cdot S^{-1}_{c \times c} \tag{1.4}$$

$\mathbf{u}_{new}$
-0.23
-0.89

$\mathbf{u}$
1
4
1
0

$\mathbf{V}_{m \times c}$	
-0.47	0.11
-0.28	-0.85
-0.47	-0.27
-0.69	0.45

$\mathbf{S^{-1}}_{c \times c}$	
0.11	0
0	0.25

FIGURE 1.2.4: Example of: $\mathbf{u}_{new}$ (inserted new user vector), $\mathbf{u}$ (user vector), $\mathbf{V}_{m \times c}$ (two right singular vectors of V), $\mathbf{S^{-1}}_{c \times c}$ (two singular values of inverse S).

In Equation 1.4, $\mathbf{u_{new}}$ denotes the mapped ratings of the test user $\mathbf{u}$, whereas $V_{m \times c}$ and $S^{-1}_{c \times c}$ are matrices derived from SVD. This $\mathbf{u_{new}}$ vector should be added in the end of the $U_{n \times c}$ matrix which is shown in Figure 1.2.3.

Notice that the inserted vector values of test user U_4 are very similar to these of U_2 after the insertion, as shown in Figure 1.2.5.

This is reasonable, because these two users have similar ratings as it is shown in Figure 1.2.1 (a) and Figure 1.2.1 (b).

-0.61	0.28
-0.29	**-0.95**
-0.74	0.14
-0.23	**-0.89**

FIGURE 1.2.5: The new $U_{n+1,c}$ matrix containing the new user (u_{new}) that we have added.

1.2.5 Other Factorization Methods

There are many methods on how to decompose a matrix in order to deal with a high-dimensional data set. Principal-Component Analysis or simply PCA is a data mining technique that replaces the high-dimensional original data by its projection onto the most important axes. It's a simple method, which is based on eigenvalues and eigenvectors of a matrix and it's quite effective.

Another useful decomposition method is UV decomposition. UV is an instance of SVD decomposition and its philosophy is that the original matrix is actually the product of two long "thin" matrices, U and V. UV's most often problem is overfitting. To address this problem, we can extend UV with L2 regularization, which is also known as Tikhonov regularization. That is, since the basic idea of the UV decomposition is to minimize an element-wise loss on the elements of the predicted/approximation matrix by optimizing the square loss, we can extend it with L2 regularization terms. After the application of a regularized optimization criterion the possible overfitting can be reduced.

Another widely known method in dimensionality reduction and data analysis is *Non-Negative Matrix Factorization* (NMF). The non-negative matrix factorization (NMF), also known as non-negative matrix approximation, is a group of algorithms in multivariate analysis and linear algebra where a matrix A is factorized into (usually) two matrices U and 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 [BBL$^+$06].

Assume that a_i,...,a_N are N non-negative input vectors and we organize them as the columns of non-negative data matrix A. Non-negative matrix factorization seeks a small set of K non-negative representative vectors v_i,...,v_K that can be non-negatively combine to approximate the input vectors a_i.

$$A \approx U \times V \tag{1.5}$$

and

$$a_i \approx \sum_{k=1}^{K} u_{kn} v_k, \quad 1 \leq n \leq N \tag{1.6}$$

where the combining coefficients u_{kn} are restricted to be non-negative.[DS05] There are several ways in which the U and V may be found. The main equation we presented before (see Eq.:1.5) is the most popular method to find U and V matrices.

Another method used for the decomposition of a matrix is the CUR decomposition[MD09], which presents good properties over SVD in some cases. As it is already described, SVD is able to reduce dimensions without losing approximation accuracy. However, many times in high-dimensional data sets, the produced SVD matrices are tend to be very dense, which makes their process a big challenge. In contrast, CUR decomposition confronts this

problem as it decomposes the original matrix into two sparse matrices C, R and only one dense matrix U, whose size is quite small and doesn't affects much the time complexity of the method. Another difference between SVD and CUR decompositions, is that CUR gives an exact decomposition no matter how large is the rank of the original matrix, whereas in SVD the k largest singular values shall be at least as great as the rank of the original matrix.

1.3 Higher Order Singular Value Decomposition (HOSVD) on Tensors

HOSVD is a generalization of singular value decomposition and has been successfully applied in several areas. In this section, we summarize the HOSVD procedure, apply HOSVD for recommendations in Social Tagging Systems (STSs), combine HOSVD with other methods and study the limitations of HOSVD.

1.3.1 From SVD to HOSVD

Formally, a *tensor* is a multi-dimensional matrix. A N-order tensor $\mathcal{A}$ is denoted as $\mathcal{A} \in \mathbb{R}^{I_1 \cdots I_N}$, with elements $a_{i_1,\ldots,i_N}$. The high-order singular value decomposition [LMV00] generalizes the SVD computation to tensors. To apply HOSVD on a 3-order tensor $\mathcal{A}$, three *matrix unfolding*[6] operations are defined as follows [LMV00]:

$$A_1 \in \mathbb{R}^{I_1 \times (I_2 I_3)}, \qquad A_2 \in \mathbb{R}^{I_2 \times (I_1 I_3)}, \qquad A_3 \in \mathbb{R}^{(I_1 I_2) \times I_3} \tag{1.1}$$

where A_1, A_2, A_3 are called the mode-1, mode-2, mode-3 matrix unfolding of $\mathcal{A}$, respectively. The unfoldings of $\mathcal{A}$ in the three modes are illustrated in Figure 1.3.6.

In the following, we will present an example of tensor decomposition adopted from [LMV00]:

Example 1.3.1 *Define a tensor* $\mathcal{A} \in \mathbb{R}^{3\times2\times3}$ *by* $a_{1,1,1} = a_{1,1,2} = a_{2,1,1} = -a_{2,1,2} = 1, a_{2,1,3} = a_{3,1,1} = a_{3,1,3} = a_{1,2,1} = a_{1,2,2} = a_{2,2,1} = -a_{2,2,2} = 2, a_{2,2,3} = a_{3,2,1} = a_{3,2,3} = 4, a_{1,1,3} = a_{3,1,2} = a_{1,2,3} = a_{3,2,2} = 0$. *The tensor and its mode-1 matrix unfolding* $A_1 \in \mathbb{R}^{I_1 \times I_2 I_3}$ *are illustrated in Figure 1.3.7.*

[6] We define as "matrix unfolding" of a given tensor the matrix representations of that tensor in which all the column (row, ...) vectors are stacked one after the other.

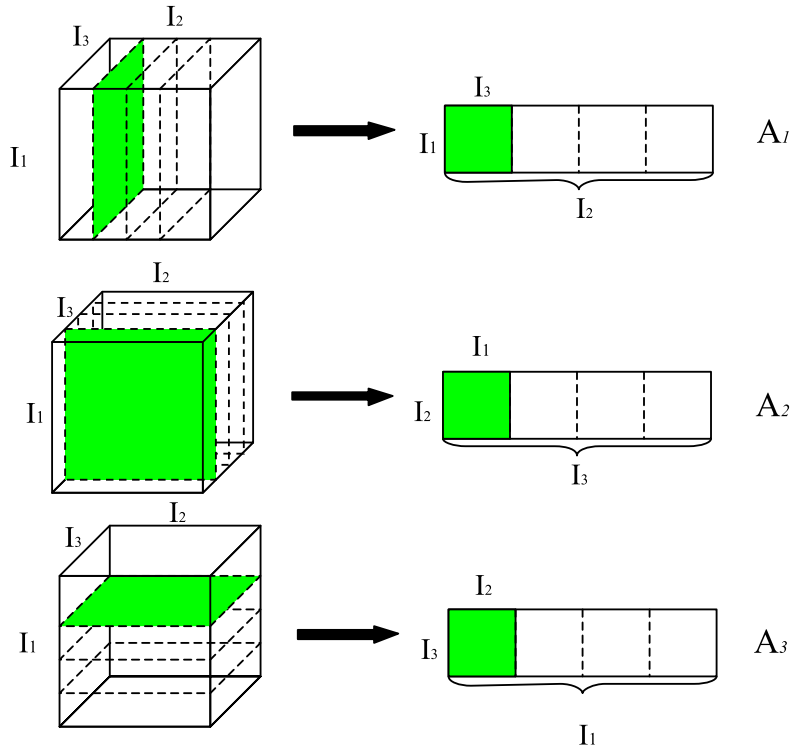


FIGURE 1.3.6: Visualization of the three unfoldings of a 3-order tensor.

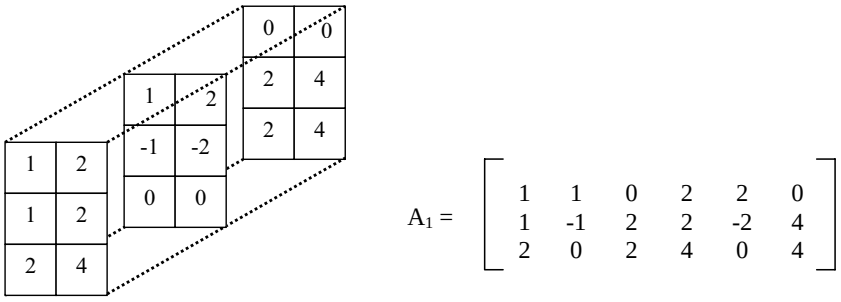


FIGURE 1.3.7: Visualization of tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{3\times 2\times 3}$ and its mode-1 matrix unfolding.

Next, we define the mode-n product of a N-order tensor $\mathcal{A} \in \mathbb{R}^{I_1\times\cdots\times I_N}$ by a matrix $U \in \mathbb{R}^{J_n\times I_n}$, which is denoted as $\mathcal{A} \times_n U$. The result of the mode-n product is an $(I_1 \times I_2 \times \cdots \times I_{n-1} \times J_n \times I_{n+1} \times \cdots \times I_N)$-tensor, the entries of which are defined as follows:

$$(\mathcal{A} \times_n U)_{i_1 i_2 \ldots i_{n-1} j_n i_{n+1} \ldots i_N} = \sum_{i_n} a_{i_1 i_2 \ldots i_{n-1} i_n i_{n+1} \ldots i_N} u_{j_n, i_n} \tag{1.2}$$

Since we focus on 3-order tensors, $n \in \{1, 2, 3\}$, we use mode-1, mode-2, and mode-3 products.

In terms of mode-n products, SVD on a regular two-dimensional matrix (i. e., 2-order tensor), can be rewritten as follows [LMV00]:

$$F = S \times_1 U^{(1)} \times_2 U^{(2)} \tag{1.3}$$

where $U^{(1)} = (u_1^{(1)} u_2^{(1)} \ldots u_{I_1}^{(1)})$ is a *unitary* $(I_1 \times I_1)$-matrix [7], $U^{(2)} = (u_1^{(2)} u_2^{(2)} \ldots u_{I_1}^{(2)})$ is a *unitary* $(I_2 \times I_2)$-matrix, and S is a $(I_1 \times I_2)$-matrix with the properties of:

i. pseudo-diagonality: $S = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_{\min\{I_1, I_2\}})$

ii. ordering: $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min\{I_1, I_2\}} \geq 0$.

By extending this form of SVD, HOSVD of a 3-order tensor $\mathcal{A}$ can be written as follows [LMV00]:

$$\mathcal{A} = S \times_1 U^{(1)} \times_2 U^{(2)} \times_3 U^{(3)} \tag{1.4}$$

where $U^{(1)}$, $U^{(2)}$, $U^{(3)}$ contain the orthonormal vectors (called the mode-1, mode-2 and mode-3 singular vectors, respectively) spanning the column space of the A_1, A_2, A_3 matrix unfoldings. S is called core tensor and has the property of "all orthogonality".[8] This decomposition also refers to a general factorization model known as Tucker decomposition [Tuc66].

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

[7] An $n \times n$ matrix U is said to be unitary if its column vectors form an orthonormal set in the complex inner product space $\mathbb{C}^n$. That is, $U^T U = I_n$.

[8] All- orthogonality means that the different "horizontal matrices" of S (the first index i_1 is kept fixed, while the two other indices, i_2 and i_3, are free) are mutually orthogonal with respect to the scalar product of matrices (i. e., the sum of the products of the corresponding entries vanishes); at the same time, the different "frontal" matrices (i_2 fixed) and the different "vertical" matrices (i_3 fixed) should be mutually orthogonal as well. For more information, see [LMV00].

SVD in each new matrix. Finally, we build the core tensor $\mathcal{S}$ and the resulting tensor $\hat{\mathcal{A}}$. The six steps of the HOSVD approach are summarized as follows:

- *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: 1.1)
- *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 the dimensionality. (see eq: 1.3)
- *Step 5*: The construction of the $\hat{\mathcal{A}}$ tensor, that is an approximation of tensor $\mathcal{A}$. (see eq: 1.4)
- *Step 6*: Based on the weights of the elements of the reconstructed tensor $\hat{\mathcal{A}}$, we recommend item/tag to the target user u.

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

1.3.2 HOSVD for Recommendations in Social Tagging Systems (STSs)

In this subsection, we elaborate on how HOSVD can be employed for computing recommendations in STS and present an example on how one can recommend items according to the detected latent associations. Although we illustrate only the recommendation of items, once the approximation $\hat{\mathcal{A}}$ is computed the recommendation of users or tags is straightforward [SNM10].

The ternary relation of users, items and tags can be represented as a third-order tensor $\mathcal{A}$, such that tensor factorization techniques can be employed in order to exploit the underlying latent semantic structure in $\mathcal{A}$. While the idea of computing low rank tensor approximations has already been used for many different purposes [LMV00, SH05, KS08, WA08, CWZ07, SZL+05], just recently it has been applied for the problem of recommendations in STS. The basic idea is to cast the recommendation problem as a third-order tensor completion problem — completing the non-observed entries in $\mathcal{A}$.

Formally, a social tagging system is defined as a relational structure $\mathbb{F} := (U, I, T, Y)$ in which

- U, I, and T are disjoint non-empty finite sets, whose elements are called users, items, and tags, respectively, and

- Y is the set of observed ternary relations between them, i. e., $Y \subseteq U \times I \times T$, whose elements are called tag assignments.
- A post corresponds to the set of tag assignments of a user for a given item, i. e., a triple $(u, i, T_{u,i})$ with $u \in U$, $i \in I$, and a non-empty set $T_{u,i} := \{t \in T \mid (u, i, t) \in Y\}$.

In the following we present several approaches for recommending in STS based on tensor factorization.

Y which represents the ternary relation of users, items and tags can be depicted by the binary tensor $\mathcal{A} = (a_{u,i,t}) \in \mathbb{R}^{|U|\times|I|\times|T|}$ where 1 indicates observed tag assignments and 0 missing values, i. e.,

$$a_{u,i,t} := \begin{cases} 1, & (u, i, t) \in Y \\ 0, & \text{else} \end{cases}$$

Now, we express the tensor decomposition as

$$\hat{\mathcal{A}} := \hat{C} \times_u \hat{U} \times_i \hat{I} \times_t \hat{T} \tag{1.5}$$

where $\hat{U}$, $\hat{I}$, and $\hat{T}$ are low-rank feature matrices representing a mode (i. e., user, items, and tags, respectively) in terms of its small number of latent dimensions k_U, k_I, k_T, and $\hat{C} \in \mathbb{R}^{k_U \times k_I \times k_T}$ is the core tensor representing interactions between the latent factors. The model parameters to be optimized are represented by the quadruple $\hat{\theta} := (\hat{C}, \hat{U}, \hat{I}, \hat{T})$ (see Figure 1.3.8).

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, i. e.,

$$\underset{\hat{\theta}}{\operatorname{argmin}} \sum_{(u,i,t)\in Y} (\hat{a}_{u,i,t} - a_{u,i,t})^2$$

After the parameters are optimized, predictions can be done as follows:

$$\hat{a}(u, i, t) := \sum_{\tilde{u}=1}^{k_U} \sum_{\tilde{i}=1}^{k_I} \sum_{\tilde{t}=1}^{k_T} \hat{c}_{\tilde{u},\tilde{i},\tilde{t}} \cdot \hat{u}_{u,\tilde{u}} \cdot \hat{i}_{i,\tilde{i}} \cdot \hat{t}_{t,\tilde{t}} \tag{1.6}$$

where $\hat{U} = [\hat{u}_{u,\tilde{u}}]^{u=1,\ldots,U}_{\tilde{u}=1,\ldots,k_U}$, $\hat{I} = [\hat{i}_{i,\tilde{i}}]^{i=1,\ldots,I}_{\tilde{i}=1,\ldots,k_I}$, $\hat{T} = [\hat{t}_{t,\tilde{t}}]^{t=1,\ldots,T}_{\tilde{t}=1,\ldots,k_T}$ and indices over the feature dimension of a feature matrix are marked with a tilde, and elements of a feature matrix are marked with a hat (e. g., $\hat{t}_{t,\tilde{t}}$).

Example 1.3.2 *The HOSVD algorithm takes $\mathcal{A}$ as input and outputs the reconstructed tensor $\hat{\mathcal{A}}$. $\hat{\mathcal{A}}$ measures the strength of associations between users, items, and tags. Each element of $\hat{\mathcal{A}}$ can be represented by a quadruplet $\{u, i, t, p\}$, where p 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 the $\{u, t\}$ pair.*

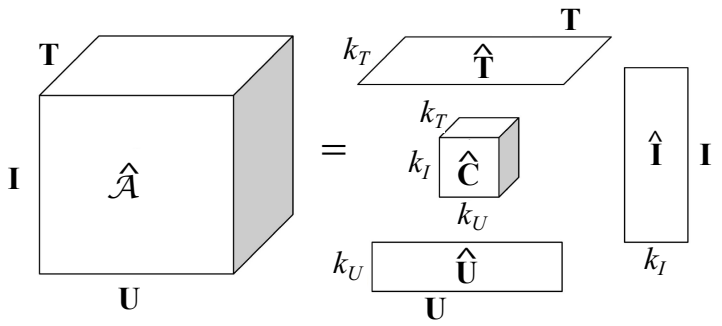


FIGURE 1.3.8: Tensor decomposition in STS. Figure adapted from [RMNST09].

In this subsection, in order to illustrate how HOSVD for item recommendation works, we apply HOSVD to a toy example. As illustrated in Figure 1.3.9, three users tagged three different items (web links). In Figure 1.3.9, 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, the annotated numbers on the arrow lines gives 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.

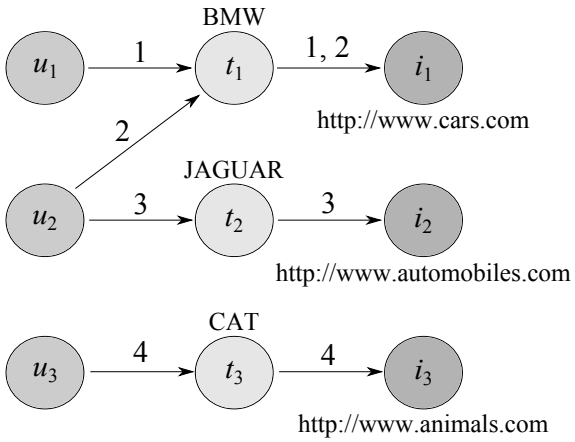


FIGURE 1.3.9: Usage data of the running example.

From Figure 1.3.9, we can see that users u_1 and u_2 have common interests on cars, while user u_3 is interested in cats. A 3-order tensor $\mathcal{A} \in \mathbb{R}^{3\times3\times3}$, can be constructed from the usage data. We use the co-occurrence frequency

(denoted as weights) of each triplet user, item, and tag as the elements of tensor $\mathcal{A}$, which are given in Table 1.3.1. Note that all associated weights are initialized to 1. Figure 1.3.10 presents the tensor construction of our running example.

TABLE 1.3.1: 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

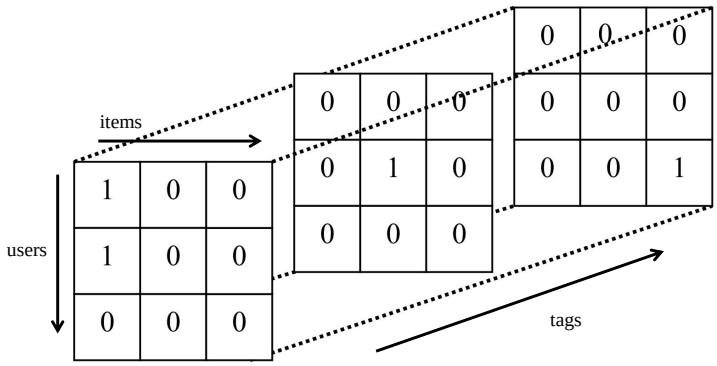


FIGURE 1.3.10: The tensor construction of our running example.

After performing the tensor reduction analysis, we can get the reconstructed tensor of $\hat{\mathcal{A}}$, which is presented in Table 1.3.2, whereas Figure 1.3.11 depicts the contents of $\hat{\mathcal{A}}$ graphically (the weights are omitted). As shown in Table 1.3.2 and Figure 1.3.11, 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. This association is represented with the last (bold faced) row in Table 1.3.2 and with the dashed arrow line in Figure 1.3.11).

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 1.3.2 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 presented in Figure 1.3.12.

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

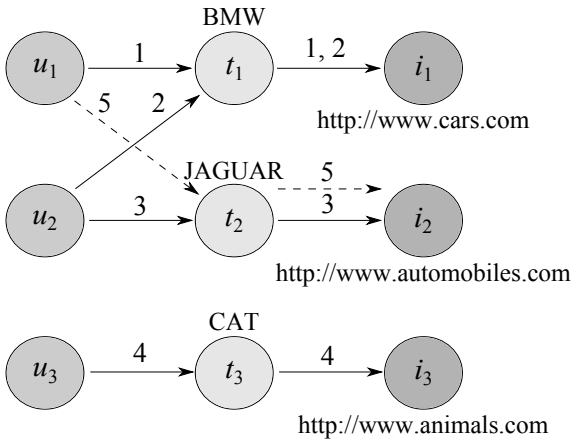


FIGURE 1.3.11: Illustration of the tensor reduction algorithm output for the running example.

TABLE 1.3.2: 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**

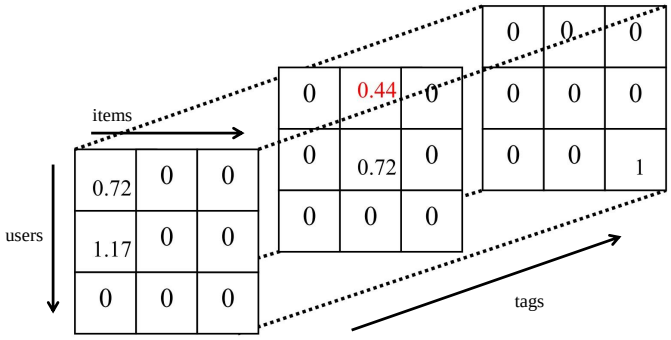


FIGURE 1.3.12: The resulting $\hat{\mathcal{A}}$ tensor for the running example.

the latent associations among the multi-type data objects: user, items, and tags. The associations can then be used to improve the item recommendation procedure.

1.3.3 Handling the Sparsity Problem

Sparsity is a severe problem in 3-dimensional data, and it can affect the outcome of SVD. To address this problem, instead of SVD we can apply kernel-SVD [CSS06, CST04] in the 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 [SNM10].

For each unfolding A_i ($1 \leq i \leq 3$) we have to non-linearly map its contents to a higher dimensional space using a mapping function ϕ. Therefore, from each A_i matrix we can 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:

$$F_i = U^{(i)} S^{(i)} (V^{(i)})^T \tag{1.7}$$

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

Nevertheless, to avoid the 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 the matrices with the left-singular vectors, for each mode i we can define a matrix B_i as follows:

$$B_i = F_i F_i^T \tag{1.8}$$

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" [CST04] 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 Equations 1.7 and 1.8 that:

$$B_i = (U^{(i)} S^{(i)} (V^{(i)})^T)(U^{(i)} S^{(i)} (V^{(i)})^T)^T = U^{(i)} (S^{(i)})^2 (V^{(i)})^T \tag{1.9}$$

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

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.

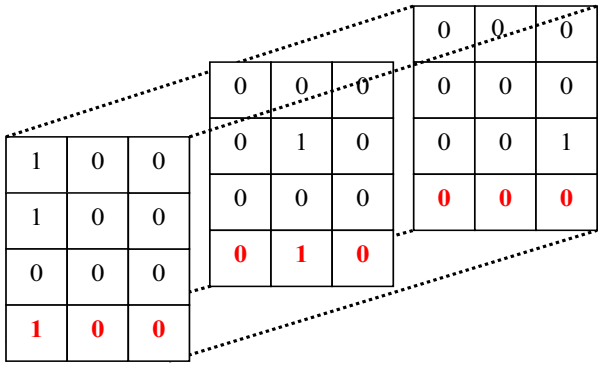
1.3.4 Inserting new users, tags, or items

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 the SVD of the corresponding mode in Equations 1.7 & 1.9. As we would like to avoid the costly batch recomputation of the corresponding SVD, we can consider incremental solutions [SKR02, Bur02]. 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 [FDD+88, SKR02], whereas for larger update sizes we can consider Incremental SVD techniques [Bur02]. Both techniques are described next [SNM10].

1.3.4.1 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 the entries of the new user result to the appending of new row in A_1. This is exemplified in Figure 1.3.13. Figure 1.3.13a shows the insertion of a new user in the tensor of the current example (the new values are presented with red color). Notice that to ease the presentation, the new user's tags and items are identical with those of user U_2.



(a)

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

(b)

FIGURE 1.3.13: Example of folding in a new user: a) the insertion of a new user in the tensor, b) the new 1-mode unfolded matrix $\mathbf{A}_1$.

Let $\mathbf{u}$ denote the new row that is appended to A_1. Figure 1.3.13b presents the new A_1, i.e., the 1-mode unfolded matrix, where it is shown that the 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 by Equation 5. To avoid batch SVD recomputation, we can use the existing basis $U_{c1}^{(1)}$ of left singular vectors, to project the $\mathbf{u}$ row onto the the reduced $c1$-dimensional

space of users in the A_1 matrix. This projection is called folding-in and is computed by using the following Equation 1.10 [FDD+88]:

$$\mathbf{u_{new}} = \mathbf{u} \cdot V_{c1}^{(1)} \cdot (S_{c1}^{(1)})^{-1} \tag{1.10}$$

In Equation 1.10, $\mathbf{u_{new}}$ denotes the mapped row, which will be appended to $U_{c1}^{(1)}$, whereas $V_{c1}^{(1)}$ and $(S_{c1}^{(1)})^{-1}$ are the dimensionally reduced matrixes derived when SVD was originally applied to A_1, i.e., before the insertion of the new user. In the current example, the computation of $\mathbf{u_{new}}$ is described in Figure 1.2.4.

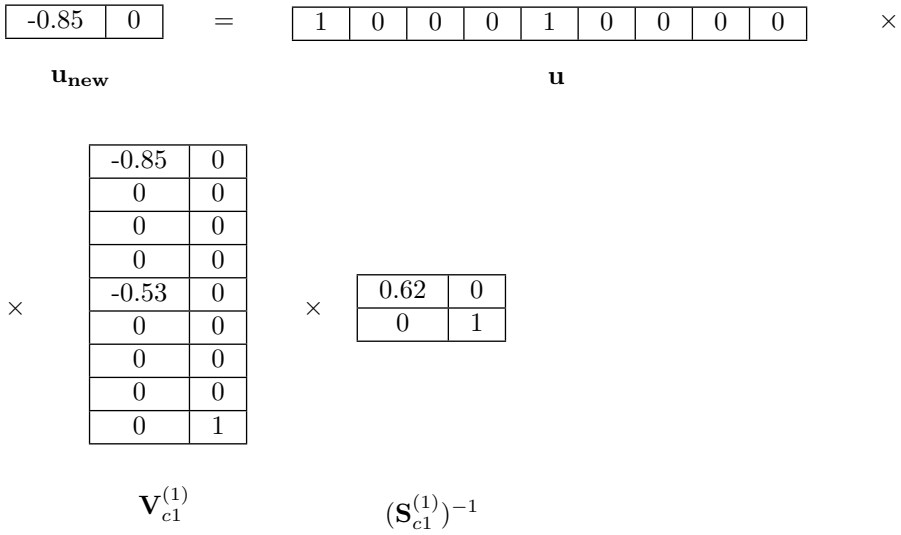


FIGURE 1.3.14: The result of folding-in for the current example.

The $\mathbf{u_{new}}$ vector should be appended in 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).

Finally, to update the tensor $\hat{\mathcal{A}}$, we have to perform the products given in Equation 1.7. Notice that only $U^{(1)_{c1}}$ has been modified in this equation. Thus, to optimize the 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 pre-stored so as to avoid its recomputation. For the current example, the resulting $\hat{\mathcal{A}}$ tensor is shown in Figure 1.3.15.

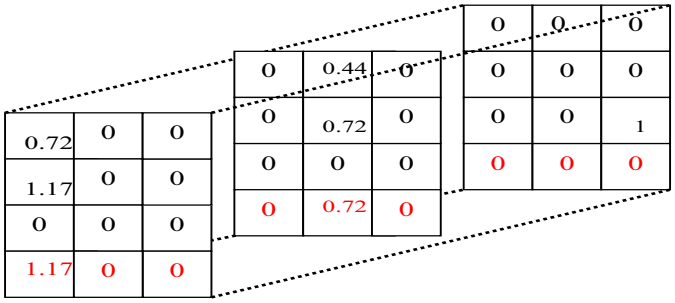


FIGURE 1.3.15: The resulting $\hat{\mathcal{A}}$ tensor of the running example after the insertion of the new user.

Analogous insertion procedure can be followed for the insertion of a new item or tag. For a new item insertion, we have to apply Equation 1.10 on the 2-mode matrix unfolding of tensor $\mathcal{A}$, while for a new tag we apply Equation 1.10 on the 3-mode matrix unfolding of tensor $\mathcal{A}$.

1.3.4.2 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 [SKR02]. When the update size is not big, loss of orthogonality may not be a severe problem in practice. Nevertheless, for larger update sizes the loss of orthogonality may result to 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 [Bur02].

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

$$M_{p\times q} = U_{p\times r} S_{r\times r} V^T_{r\times q} \tag{1.11}$$

Assume that each column of matrix $C_{p\times c}$ contains the 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 the subspace orthogonal to U. Consider the following identity:

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

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

$$Q = U'S'(V')^T \tag{1.12}$$

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

$$U'' = [U\ J]U',\ S'' = S',\ V'' = \begin{bmatrix} V & 0 \\ 0 & I \end{bmatrix} V' \tag{1.13}$$

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

$$[M\ C] = [USV^T\ C] = U''S''(V'')^T \tag{1.14}$$

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

Returning to the application of incremental update for new users, items, or tags, as described in Section 1.3.4.1, in each case we result with a number of new rows that are appended in 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''.

1.3.5 Other Scalable Factorization Models

The HOSVD approach has two important drawbacks:

Modeling: The runtime complexity is cubic in the size of the latent dimensions. This can be seen in Equation 1.6, where three nested sums have to be calculated just for predicting a single (user, item, tag)-triple. There are several approaches to improve the efficiency of HOSVD [KS08, Tur07, DM07].

Learning: HOSVD is optimized for least-squares on the whole tensor $\mathcal{A}$. However, recommendation is a ranking task not a regression task and also the non-observed posts are not taken into account by HOSVD.

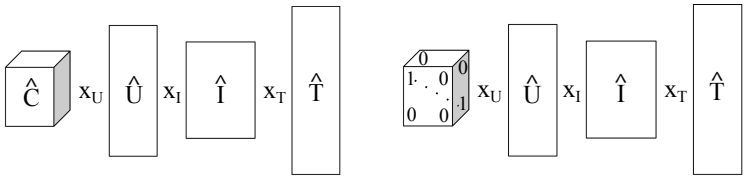
We will study both issues next.

The limitation in runtime of HOSVD stems from its model which is the Tucker Decomposition. In the following, we will discuss a second factorization model (i.e., PARAFAC) that has been proposed for tag recommendation. We investigate its model assumptions, complexity and its relation with HOSVD.

The underlying tensor factorization model of HOSVD is the Tucker Decomposition (TD) [Tuc66]. As noted before, for tag recommendation, the model reads:

$$\hat{\mathcal{A}} := \hat{C} \times_u \hat{U} \times_i \hat{I} \times_t \hat{T} \tag{1.15}$$

The reason for the cubic complexity (i. e., $O(k^3)$ with $k := \min(k_U, k_I, k_T)$) of TD is the core tensor.



(a) TD (b) PARAFAC

FIGURE 1.3.16: Relationship between Tucker Decomposition and Parallel Factor Analysis (PARAFAC).

The Parallel Factor Analysis (PARAFAC) [Har70] model aka canonical decomposition [CC70] reduces the complexity of the TD model by assuming a diagonal core tensor.

$$c_{\tilde{u},\tilde{i},\tilde{t}} \stackrel{!}{=} \begin{cases} 1, & \text{if } \tilde{u} = \tilde{i} = \tilde{t} \\ 0, & \text{else} \end{cases} \tag{1.16}$$

which allows to rewrite the model equation:

$$\hat{a}_{u,i,t} = \sum_{f=1}^{k} \hat{u}_{u,f} \cdot \hat{i}_{i,f} \cdot \hat{t}_{t,f} \tag{1.17}$$

In contrast to TD, the model equation of PARAFAC can be computed in $O(k)$. In total, the model parameters $\hat{\theta}$ of the PARAFAC model are:

$$\hat{U} \in \mathbb{R}^{|U| \times k}, \quad \hat{I} \in \mathbb{R}^{|I| \times k}, \quad \hat{T} \in \mathbb{R}^{|T| \times k} \tag{1.18}$$

The assumption of a diagonal core tensor is a restriction of the TD model.

A graphical representation of Tucker Decomposition (TD) and Parallel Factor Analysis (PARAFAC) shown in Figure 1.3.16. It can be seen that any PARAFAC model can be expressed by a TD model (with diagonal core tensor).

Let $\mathcal{M}$ be the set of models that can be represented by a model class. In [Ren10] it is shown that for tag recommendation

$$\mathcal{M}^{\text{TD}} \supset \mathcal{M}^{\text{PARAFAC}} \tag{1.19}$$

This means that any PARAFAC model can be expressed with a TD model but there are TD models that cannot be represented with a PARAFAC model. In [RST10, Ren10] it was pointed out that this does not mean that TD is guaranteed to have a higher prediction quality than PARAFAC. On the contrary, as all model parameters are estimated from limited data, restricting the expressiveness of a model can lead to higher prediction quality if the restriction is in line with the true parameters.

1.4 A Real Geo-Social System based on HOSVD

This section presents a real-world recommender system for Location-Based Social Networks (LBSNs)[9]. GeoSocialRec allows to test, evaluate and compare different recommendation styles in an online setting, where the users of GeoSocialRec actually receive recommendations during their check-in process.

The GeoSocialRec recommender system consists of several components. The system's architecture is illustrated in Figure 1.4.17, where three main sub-systems are described: (i) the Web Site, (ii) the Database Profiles and (iii) the Recommendation Engine. In the following sections, we describe each sub-system of GeoSocialRec in detail.

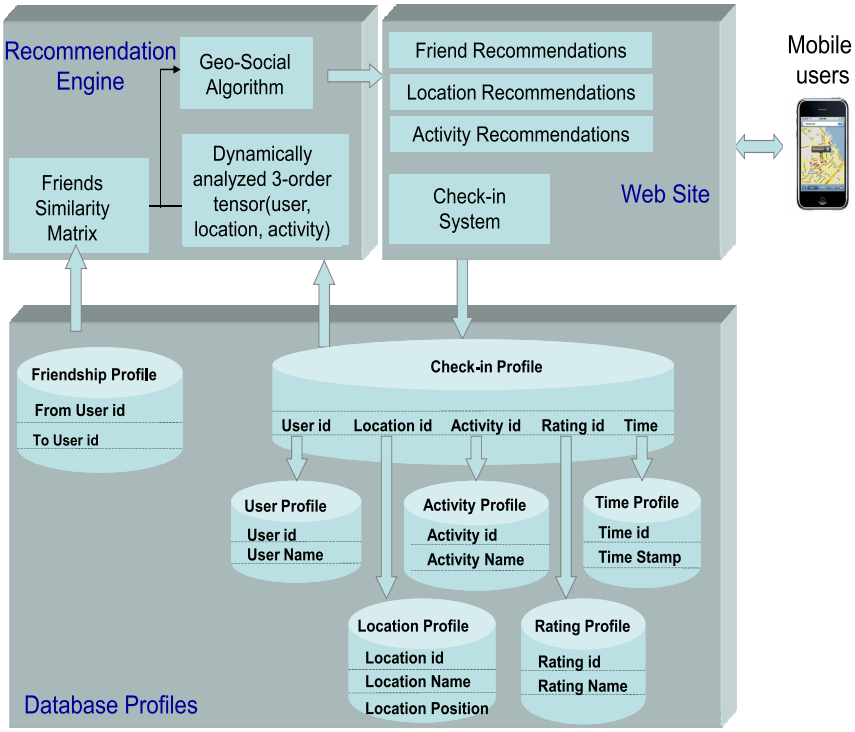


FIGURE 1.4.17: Components of the Geo-social recommender system.

[9] *"A LSBN does not only mean adding a location to an existing social network so that people in the social structure can share location-embedded information, but also consists of new social structure made up of individuals connected by the interdependency derived from their locations in the physical world as well as their location-tagged media content, such as photos, videos and texts"*, Microsoft Research: Location-Based Social Networks, www.research.microsoft.com

1.4.1 GeoSocialRec Web Site

The GeoSocialRec system uses a web site [10] to interact with the users. The web site consists of four subsystems: (i) the friend recommendation, (ii) the location recommendation, (iii) the activity recommendation, and (iv) the check-in subsystem. The friend recommendation subsystem is responsible for evaluating incoming data from the Recommendation Engine of GeoSocialRec and providing updated friend recommendations. We provide friend, location and activity recommendations, where new and updated location and activity recommendations are presented to the user as new check-ins are stored in the Database profiles. Finally, the check-in subsystem is responsible for passing the data inserted by the users to the respective Database profiles.

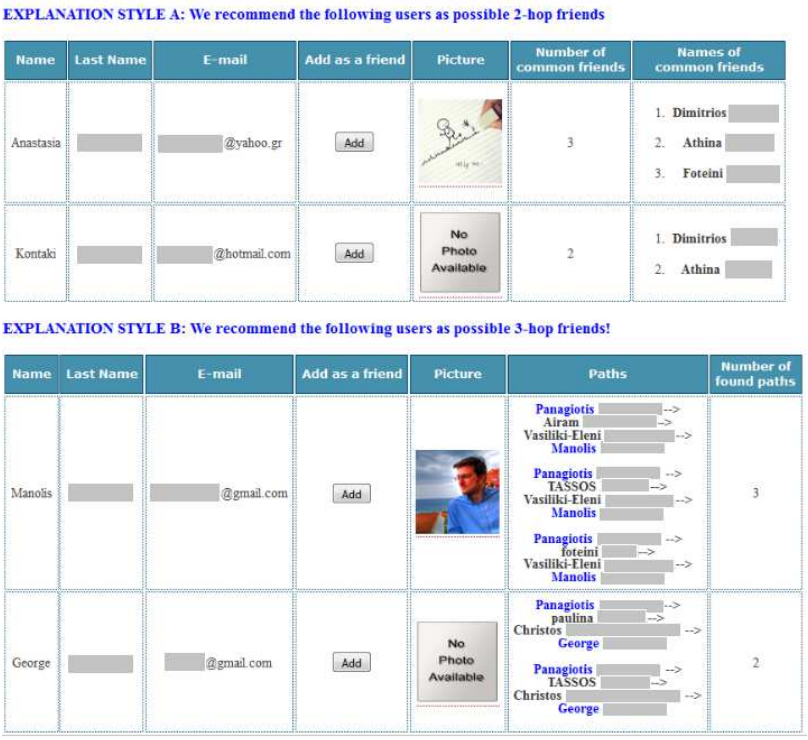


FIGURE 1.4.18: Friend recommendations provided by the GeoSocialRec system.

Figure 1.4.18 presents a scenario where the GeoSocialRec system recommends four possible friends to user *Panagiotis Symeonidis*. As shown, the first table recommends Anastasia Kalou and Ioanna Kontaki, who are connected to him with 2-hop paths. The results are ordered based on the second to

[10] http://delab.csd.auth.gr/geosocialrec

last column of the table, which indicates the number of common friends that the target user shares with each possible friend. As shown in Figure 1.4.18, Anastasia Kalou is the top recommendation because she shares 3 common friends with the target user. The common friends are then presented in the last column of the table. The second table contains two users, namely Manolis Daskalakis and George Tsalikidis, who are connected to the target user via 3-hop paths. The last column of the second table indicates the number of found paths that connect the target user with the recommended friends. Manolis Daskalakis is now the top recommendation, because he is connected to Panagiotis Symeonidis via three 3-hop paths. It is obvious that the second explanation style is more analytical and detailed, since users can see, in a transparent way, the paths that connect them with the recommended friends.

Figure 1.4.19a shows a location recommendation, while Figure 1.4.19b depicts an activity recommendation. As shown in Figure 1.4.19a, the target user can provide to the system the activity she wants to do and the place she is (i.e. Bar in Athens). Then, the system provides a map with bar places (i.e. place A, place B, place C, etc.) along with a table, where these places are ranked based on the number of users' check-ins and their average rating. As shown in Figure 1.4.19a, the top recommended Bar is Mojo (i.e. place A), which is visited 3 times (from the target user's friends) and is rated highly (i.e. 5 stars). Regarding the activity recommendation, as shown in Figure 1.4.19b, the user selects a nearby city (i.e. Thessaloniki) and the system provides activities that she could perform. In this case, the top recommended activity is sightseeing the White Tower of Thessaloniki, because it is visited 14 times and has an average rating of 4.36.

1.4.2 GeoSocialRec Database and Recommendation Engine

The database that supports the GeoSocialRec system is a MySQL(v.5.5.8) [11] database. MySQL is an established Database Management System (DBMS), which is widely used in on-line, dynamic, database driven websites.

The database profile sub-system contains five profiles where data about the users, locations, activities and their corresponding ratings are stored. As shown in Figure 1.4.17, this data are received by the Check-In profile and along with the Friendship profile, they provide the input for the Recommendation Engine sub-system. Each table field represents the respective data that is collected by the Check-In profile. User-id, Location-id and Activity-id refer to specific ids given to users, locations and activities respectively.

The recommendation engine is responsible for collecting the data from the database and producing the recommendations, which will then be displayed on the web site. As shown in Figure 1.4.17, the recommendation engine constructs a friends similarity matrix based on FriendLink [APM11] algorithm. The average geographical distances (*in kilometres*) between users' check-ins

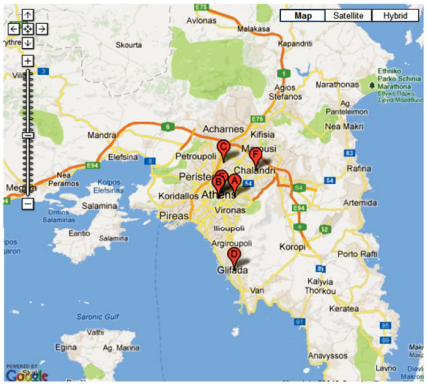
[11] http://www.mysql.com

We recommend the following Point(s) of Interest
for the Activity: Bar
in the city of:Athens

EXPLANATION STYLE A:

We recommend the following POI's (Point of Interest) based on total Check-ins!

A/A	Point Of Interest	POI Address	Explanation Style A: Total Check-Ins	Average Rating from style A	Go To
A	Mojo	Παπαδιαμαντοπούλου 8, Ζωγράφου 157 71, Ελλάδα	3	5.0000	Move!
B	A for Athens	Ερμού 82, Αθήνα 105 55, Ελλάδα	3	3.6667	Move!
C	Rox Box	Ειρήνης 2-10, Φιλαδέλφεια Χαλκηδόνα 143 41, Ελλάδα	2	4.5000	Move!
D	Holy Spirit	Λαοδικης 18, Γλυφάδα 166 74, Ελλάδα	2	4.0000	Move!
E	Mo Better	Κωλέττη 28-42, Αθήνα, Ελλάδα	2	4.0000	Move!
F	Allo Bar	Thoukydidou 9-13, Chalandri 15232, Greece	2	2.0000	Move!



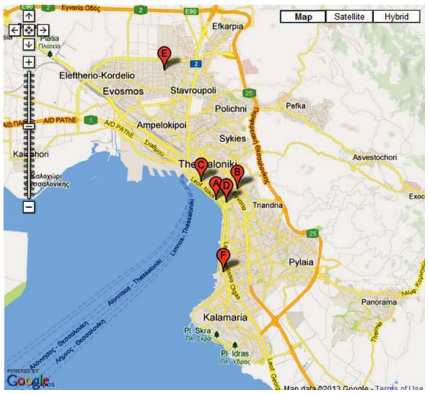
(a)

We recommend the following Activities
in the city of:Thessaloniki

EXPLANATION STYLE A:

We recommend the following activities based on total Check-ins!

A/A	Activity	Point Of Interest	POI Address	Explanation Style A: Total Check-Ins	Average Rating from style A	Go To
A	Sight-seeing	White Tower	Nikis Avenue--Paralia Thessalonikis	14	4.3571	Move!
B	Education	Aristotle University of Thessaloniki	Egnatia & Kondriktonos--Aristotle Campus	13	4.2308	Move!
C	Sight-seeing	Aristotelous Square	Aristotle Square--City Center	11	4.1818	Move!
D	Museums	Archaeological Museum of Thessaloniki	Ανδρόνικου, Θεσσαλονίκη 54621, Ελλάς--	8	4.0000	Move!
E	Cinema	Video Land	Ιθάκης 63-71, Εύοσμο 56224, Ελλάς--	8	2.6250	Move!
F	Bar	BRISTOL	George Papandreou 24--Poseidonio	7	4.2857	Move!



(b)

FIGURE 1.4.19: Location and activity recommendations made by the Geo-social recommender system.

are used as link weights. To obtain the weights, we calculate the average distance between all pairs of Points Of Interests (POIs) that two users have checked-in. The recommendation engine also produces a dynamically analyzed 3-order tensor, which is firstly constructed by the HOSVD algorithm and is

then updated using incremental methods [Bra02, SKKR02], both of which are explained previously in Sections 1.2.4 and 1.3.4, respectively.

1.4.3 Experiments

In this section, we study the performance of FriendLink [APM11] algorithm and HOSVD method in terms of friend, location and activity recommendations. To evaluate the aforementioned recommendations we have chosen two real data sets. The first one, denoted as GeoSocialRec data set, is extracted from the GeoSocialRec site [12]. It consists of 102 users, 46 locations and 18 activities. The second data set, denoted as UCLAF [ZCZ+10], consists of 164 users, 168 locations and 5 different types of activities, including "Food and Drink", "Shopping", "Movies and Shows", "Sports and Exercise", and "Tourism and Amusement".

The numbers c_1, c_2, and c_3 of left singular vectors of matrices $U^{(1)}$, $U^{(2)}$, $U^{(3)}$ for HOSVD, after appropriate tuning, are set to 25, 12 and 8 for the GeoSocialRec dataset, and to 40, 35, 5 for the UCLAF data set. Due to lack of space we do not present experiments for the tuning of c_1, c_2, and c_3 parameters. The core tensor dimensions are fixed, based on the aforementioned c_1, c_2, and c_3 values.

We perform 4-fold cross validation and the default size of the training set is 75% (we pick, for each user, 75% of his check-ins and friends randomly). The task of all three recommendation types (i.e. friend, location, activity) is to predict the friends/locations/activities of the user's 25% remaining check-ins and friends, respectively. As performance measures we use precision and recall, which are standard in such scenarios. For a test user that receives a list of N recommended friends/locations/activities

Next, we study the accuracy performance of HOSVD in terms of precision and recall. This reveals the robustness of HOSVD in attaining high recall with minimal losses in terms of precision. We examine the top-N ranked list, which is recommended to a test user, starting from the top friend/location/activity. In this situation, the recall and precision vary as we proceed with the examination of the top-N list. In Figure 1.4.20, we plot a precision versus recall curve.

As it can be seen, the HOSVD approach presents high accuracy. The reason is that we exploit altogether the information that concerns the three entities (friends, locations, and activities) and thus, we are able to provide accurate location/activity recommendations. Notice that activity recommendations are more accurate than location recommendations. A possible explanation could be the fact that the number of locations is bigger than the number of activities. That is, it is easier to predict accurately an activity than a location. Notice that for the task of friend recommendation, the performance of Friendlink is not so high. The main reason is data sparsity. In particular, the friendship

[12] http://delab.csd.auth.gr/\~symeon

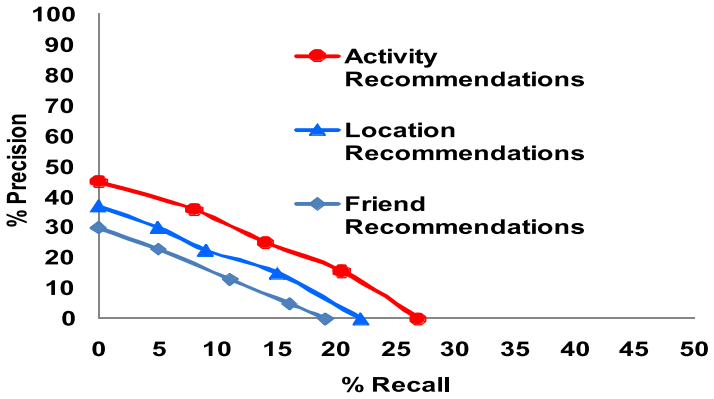


FIGURE 1.4.20: Precision-recall diagram of HOSVD and FriendLink for activity, location and friend recommendations on the GeoSocialRec data set.

network has average nodes' degree equal to 2.7 and average shortest distance between nodes 4.7, which means that the friendship network can not be consider as a "small world" network and friend recommendations can not be so accurate.

For the UCLAF data set, as shown in Figure 1.4.21, the HOSVD algorithm attains analogous results. Notice that the recall for the activity recommendations, reaches 100% because the total number of activities is 5. Moreover, notice that in this diagram, we do not present results for the friend recommendation task, since there is no friendship network in the corresponding UCLAF data set.

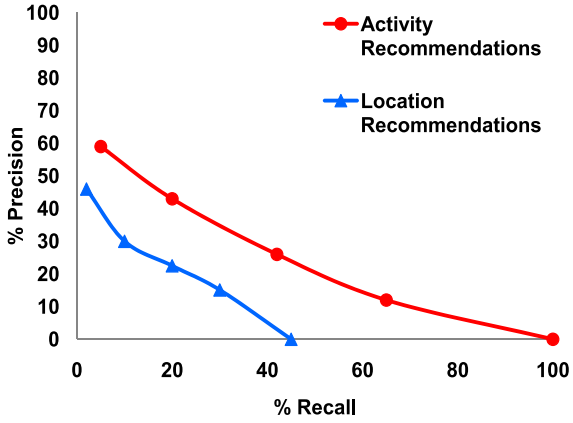


FIGURE 1.4.21: Precision-recall diagram of HOSVD for activity and location recommendations on the UCLAF data set.

1.5 Conclusion

In this chapter, we described matrix and tensor factorization techniques in recommender systems and social tagging systems, respectively. In addition, we presented a real-world recommender system for location-based social networks, which employs tensor decomposition techniques. As shown, matrix and tensor decompositions are suitable for scenarios in which the data is extremely large, very sparse, and too noisy, since the reduced representation of the data can be interpreted as a de-noisified approximation of the "true" data.

Bibliography

[APM11] P. S. A. Papadimitriou and Y. Manolopoulos. Friendlink: Link prediction in social networks via bounded local path traversal. In *In Proceedings of the 3rd Conference on Computational Aspects of Social Networks (CASON'2011)*, 2011.

[BBL+06] 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.

[Bra02] M. Brand. Incremental singular value decomposition of uncertain data with missing values. In *Proceedings of the 7th European Conference on Computer Vision (ECCV)*, pages 707–720, Copenhagen, Denmark, 2002.

[Bur02] R. Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, 2002.

[CC70] J. Carroll and J. Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of eckart-young decomposition. *Psychometrika*, 35:283–319, 1970.

[CSS06] T. Chin, K. Schindler, and D. Suter. Incremental kernel svd for face recognition with image sets. In *In Proc. FGR Conf.*, pages 461–466, 2006.

[CST04] N. Cristianini and J. Shawe-Taylor. Kernel methods for pattern analysis. *Cambridge University Press*, 2004.

[CWZ07] S. Chen, F. Wang, and C. Zhang. Simultaneous heterogeneous data clustering based on higher order relationships. In *Proceedings of the Seventh IEEE International Conference on Data Mining Workshops*, ICDMW '07, pages 387–392, Washington, DC, USA, 2007. IEEE Computer Society.

[DDF+90] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[DM07] P. Drineas and M. W. Mahoney. A randomized algorithm for a tensor-based generalization of the svd. *Linear Algebra and Its Applications*, 420(2–3):553–571, 2007.

[DS05] I. S. Dhillon and S. Sra. Generalized nonnegative matrix approximations with bregman divergences. In *In: Neural Information Proc. Systems*, pages 283–290, 2005.

[FDD+88] G. W. Furnas, S. Deerwester, S. T. Dumais, T. K. Landauer, R. A. Harshman, L. A. Streeter, and K. E. Lochbaum. Information retrieval using a singular value decomposition model of latent semantic structure. In *Proceedings of the 11th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '88, pages 465–480, New York, NY, USA, 1988. ACM.

[Har70] R. Harshman. Foundations of the parafac procedure: models and conditions for an 'exploratory' multimodal factor analysis. In *UCLA Working Papers in Phonetics*, pages 1–84, 1970.

[Kor08] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD '08: Proceeding of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 426–434. ACM, 2008.

[Kor09] Y. Koren. Collaborative filtering with temporal dynamics. In *KDD '09: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 447–456. ACM, 2009.

[KS08] T. G. Kolda and J. Sun. Scalable tensor decompositions for multi-aspect data mining. In *ICDM '08: Proceedings of the 8th IEEE International Conference on Data Mining*, pages 363–372. IEEE Computer Society, December 2008.

[LMV00] L. D. Lathauwer, B. D. Moor, and J. Vandewalle. A multilinear singular value decomposition. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1253–1278, 2000.

[MD09] M. W. Mahoney and P. Drineas. Cur matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences*, 106(3):697–702, 2009.

[Ren10] S. Rendle. *Context-Aware Ranking with Factorization Models*. Springer Berlin Heidelberg, 1st edition, November 2010.

[RMNST09] S. Rendle, L. B. Marinho, A. Nanopoulos, and L. Schimdt-Thieme. Learning optimal ranking with tensor factorization for tag recommendation. In *KDD '09: Proceedings of the 15th ACM*

SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 727–736. ACM, 2009.

[RST10] S. Rendle and L. Schmidt-Thieme. Pairwise interaction tensor factorization for personalized tag recommendation. In *WSDM '10: Proceedings of the Third ACM International Conference on Web Search and Data Mining*. ACM, 2010.

[SH05] A. Shashua and T. Hazan. Non-negative tensor factorization with applications to statistics and computer vision. In *ICML '05: Proceedings of the 22nd International Conference on Machine Learning*, pages 792–799. ACM, 2005.

[SKKR00] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Application of dimensionality reduction in recommender system - a case study. In *Proceedings of the ACM SIGKDD Workshop on Web Mining for E-Commerce - Challenges and Opportunities (WEBKDD)*, Boston, MA, 2000.

[SKKR02] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Incremental singular value decomposition algorithms for highly scalable recommender systems. In *Proceedings 5th International Conference on Computer and Information Technology (ICCIT)*, pages 27–28, Dhaka, Bangladesh, 2002.

[SKR02] B. Sarwar, J. Konstan, and J. Riedl. Incremental singular value decomposition algorithms for highly scalable recommender systems. In *International Conference on Computer and Information Science*, 2002.

[SM08] R. Salakhutdinov and A. Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *ICML '08: Proceedings of the 25th International Conference on Machine Learning*, pages 880–887. ACM, 2008.

[SNM10] P. Symeonidis, A. Nanopoulos, and Y. Manolopoulos. A unified framework for providing recommendations in social tagging systems based on ternary semantic analysis. *IEEE Transactions on Knowledge and Data Engineering*, 22(2), 2010.

[SNPM06] P. Symeonidis, A. Nanopoulos, A. Papadopoulos, and Y. Manolopoulos. Collaborative filtering based on users trends. In *Proceedings of the 30th Conference of the German Classification Society (GfKl'2006)*, Berlin, 2006.

[Str06] G. Strang. *Linear Algebra and Its Applications*. Thomson Brooks/Cole, 2006.

[Sym07] P. Symeonidis. Content-based dimensionality reduction for recommender systems. In *Proceedings of the 31st Conference of the German Classification Society (GfKl'2007)*, Freiburg, 2007.

[SZL+05] J.-T. Sun, H.-J. Zeng, H. Liu, Y. Lu, and Z. Chen. Cubesvd: a novel approach to personalized web search. In *Proceedings of the 14th international conference on World Wide Web*, WWW '05, pages 382–390, New York, NY, USA, 2005. ACM.

[Tuc66] L. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, pages 279–311, 1966.

[Tur07] P. Turney. Empirical evaluation of four tensor decomposition algorithms. *Technical Report (NRC/ERB-1152)*, 2007.

[WA08] H. Wang and N. Ahuja. A tensor approximation approach to dimensionality reduction. *Int. J. Comput. Vision*, 76:217–229, March 2008.

[ZCZ+10] V. Zheng, B. Cao, Y. Zheng, X. Xie, and Q. Yang. Collaborative filtering meets mobile recommendation: A user-centered approach. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI)*, pages 236–241, Atlanta, GA, 2010.

Index

K-way partitioning, 303
δ-regular hypergraph, 287
κ-uniform hypergraph, 278, 284
    adjacency hypermatrix, 285
    adjacency hypermatrix
        H-eigenpair, 286
    adjacency hypermatrix
        Z-eigenpair, 287
    degree hypermatrix, 287
    Laplacian hypermatrix, 287
    unsigned Laplacian
        hypermatrix, 287

adjacency hypermatrix
    for bibliometric data, 312
adjacency matrix, 283
agglomerative clustering, 67
algebraic connectivity, 56
alpha-centrality, 111
ALS, 45
Alternating Directions Method, 309
Alternating Least Squares, 45, 314
anchor tag, 100
anchor text, 100
Apache
    Hadoop, 19
    Spark, 20
    Storm, 20
approximate isolation principle, 124
arbitrary hypergraphs, 279
attribute augmented graph, 455, 456
attributed graph, 28
authority matrix, 312

bag of words, 96, 418
basis images, 38
basis vectors, 38
Baum-Eagon inequality, 288
Baum-Eagon theorem, 279, 290
betweenness, 67, 108
BFS, 76
bibliometrics, 312
Big Data
    ADMM, 353, 367
    Association networks, 385
    Cascades, 383
    Centrality, 355
    Closeness centrality, 359
    Community identification, 373
    Contagions, 383
    Dictionary learning, 369
    Endogenous factors, 385
    Exogenous factors, 385
    Graph anomaly detection, 363
    Graph Laplacian matrix, 370
    In-network processing, 365
    ISTA, 389
    Kernel K-means, 375
    Local linear embedding, 355
    MapReduce, 377
    Mental map, 361
    Missing data, 369
    Multidimensional scaling, 355
    Network-process prediction, 372
    Nuclear norm, 364
    Nyström method, 377
    Online learning, 353
    Random projections, 377
    Random sampling, 377
    Random sketching, 378
    Randomized factorizations, 280
    Robust kernel PCA, 380
    Robust PCA, 364
    Semi-supervised learning, 369

SEMs, 385
Sketching and Validation, 378
SkeVa, 378
Sparse representation, 370
Spectral clustering, 374
Tomographic inference, 385
bilateral filters, 524
bipartite, 29
blind relevance feedback, 104
Bonacich centrality, 111
boolean retrieval model, 99
Breadth First Search, 76

CANDECOMP, 44
canonical decomposition, 44
Canonical Decomposition/Parallel Factor Analysis, 314
centrality, 59, 106
centralization, 106
Cholesky decomposition, 35
Cholesky factorization, 35
Cholesky triangle, 35
circulant hypergraph, 287
Clique Percolation Method, 71
closeness, 107
cluster, 288
external criterion, 288
internal criterion, 288
clustering coefficient, 61
collapsed Gibbs sampling (CGS), 430
community detection, 59, 412
compact support, 514
complete graph, 28
Connected Iterative Scan, 72
content link clustering, 133
cosine similarity, 99
CPM, 71
cross entropy, 420
CUR approximation, 37

DAG, 20
DBMS architectures
adjacency matrix, 487
horizontal scaling, 482, 497
index-free adjacency, 487
path index, 488
structure index, 487
DCT, 68
dead ends, 113
degree centrality, 107
Depth First Search, 76
determinant, 33
DFS, 76
diffusion kernel, 30
digraph, 27
directed, 27
Directed Acyclic Graph, 20
directed links, 312
Dirichlet prior, 429
Discrete Cosine Transform, 68
divisive clustering, 67
dominant heavy link, 78
doubly semi-stochastic, 318
Dryad, 20
dual of hypergraph, 282

egonet, 80
eigendecomposition, 33, 55
eigenvalue, 33, 55
algebraic multiplicity, 55
eigenvalue decomposition, 514
eigenvector, 33, 55
eigenvector centrality, 110
EM algorithm, 419
Erdös-Rényi method, 61
Euclidean distance, 38
Evidence Lower BOund (ELBO), 430
expanded query, 101

feature matching, 317
Fiedler vector, 70
frequent subgraph mining, 76
frequent subtree mining, 76
Frobenius norm, 32, 420
FSM, 76
full rank, 32
fuzzy incidence matrix, 327

game-theoretic equilibrium, 278
Gamma function, 429

Generalized Expectation Maximization, 421
geo-location prediction, 322
GeoNames, 324
GFS, 19
Good-Turing estimate, 427
Google
    File System, 19
Google matrix, 312
GPUs, 449
graph, 27
    adjacency matrix, 295
    cover time, 76
    incidence matrix, 295
    Laplacian, 295
    normalized Laplacian, 295
    scale-free, 61
    unnormalized Laplacian, 295
graph anomaly detection, 59
Graph clustering, 455
    Inc-Cluster algorithm, 455, 459
    SA-Cluster algorithm, 455, 457
graph clustering, 58
graph cut, 66
graph database, 477
    dynamic graph, 497
    graph DBMS, 477
    graph-enabled database, 477
    native graph database, 477
graph databases
    graph database schema, 483
graph Fourier transform, 515
graph matching, 58, 317
graph querying, 478
    conjunctive query, 479
    conjunctive regular path query, 480
    regular path query, 480
    similarity subgraph matching, 479
    subgraph query, 479
graph spectrum, 55
graph-frequencies, 514
graph-signal, 512
Group Lasso regularization, 308
Hadamard product, 31, 315
Hadoop, 19
    File System, 19
HDFS, 20
heat kernel, 30
heavy vicinity, 78
heavy-tailed distribution, 61
heterogeneous information network, 126
Higher Order SVD, 45
HITS, 115, 312
homogeneous polynomial, 279, 285
hub matrix, 312
hubs and authorities, 115
hyperedge, 281
hyperedge degree, 283
hyperedges, 30, 278
hypergraph, 30, 281
    incidence matrix construction, 320, 322, 330
hypergraph adjacency matrix, 30
hypergraph anti-rank, 283
hypergraph clustering game, 288
    payoff function, 288
    evolutionary stable strategy, 289
    expected payoff, 289
    growth transformation, 290
    Nash equilibrium, 289
    players, 288
    strategy profiles, 288
hypergraph Laplacian, 279, 301
hypergraph matching, 317
hypergraph normalized cut criterion, 279, 299
hypergraph rank, 283
hypergraph soft clustering, 304
hypergraph to graph transformation, 296
    (Zhou) normalized hypergraph Laplacian, 302
    Bolla Laplacian, 299
    clique expansion, 296
    hyperedge expansion, 304
    Rodriguez Laplacian, 299
    star expansion, 297

vertex expansion, 304
hyperlink structure, 311
hyperlinks, 94, 100
hypermatrices, 412, 438
hypermatrix, 278, 284
hypermatrix sketches, 338
hyperpath, 282

identity matrix, 29
idf, 97
image annotation, 329
image tagging, 322
implicit relevance feedback, 104
in links, 94
in-degree, 27
incidence matrix, 282
incidence vector/matrix, 435
incident to, 30
index terms, 99
indirect relevance feedback, 104
Infomap, 73
information centrality, 109
inverse document frequency, 97
inverse matrix, 33
inverted index, 96
invertible, 32
isolates, 27
Iterative Scan, 72

KASP algorithm, 449
Katz centrality, 111
Khatri-Rao product, 32, 315
Kronecker multiplication, 62
Kronecker product, 32, 315
Kullback-Leibler divergence, 38, 420

label propagation, 413, 454
Incremental LPA (ILPA), 454
Label Propagation Algorithm (LPA), 454
labeled multigraph, 30
Lanczos/Arnoldi factorization
ARPACK, 451
Laplace-Beltrami operator, 295
Laplacian regularization, 371
Latent Dirichlet Allocation (LDA), 102, 123, 413, 428
Approximate Distributed LDA (AD-LDA), 447
Async-LDA, 447
Dirichlet Compound Multinomial LDA (DCM-LDA), 447
incremental LDA, 432
mini-batch online VB-LDA, 432
MPI-PLDA, 448
o-LDA, 432
online LDA, 431
Online VB for LDA, 431
online VB-LDA, 432
PLDA, 448
PLDA+, 449
latent semantic analysis, 316
Latent Semantic Analysis (LSA), 412, 414
folding-in, 415
folding-up, 417
Incremental LSI (ILSI), 417
QR decomposition, 415
recomputing the SVD, 415
SVD updating, 415
Latent Semantic Indexing (LSI), 97, 102
Latent variable models, 414
least squares solution, 35
lexical ambiguity, 101
link analysis, 105
link farm, 124
link graph, 119
link spamming, 124
link-based topic affinity, 122
low-rank approximation, 334

MACH-HOSVD, 339
MapReduce, 446
Markov Chain Monte Carlo (MCMC), 430
matricization, 42
matrix product, 31
matrix rank, 32

maximum rank, 44
Message Passing Interface, 19
Message Passing Interface (MPI), 446
mixing rate, 76
mode, 40
Moore-Penrose pseudoinverse, 35
MPI, 19
multi-hypergraph, 281
multimedia social search, 277
multiplicative update rules, 39
multiset, 281
music recommendation, 319
music tagging, 319

natural vibration modes, 514
near clique, 78
near star, 78
NetClus, 131
network centralization, 109
NMF, 38
Non-negative Matrix Factorization, 38
Non-Negative Tensor Factorization, 46
non-singular, 32
normalized graph Laplacian, 434
normalized Laplacian, 434
NTF, 46
null space, 32
Nyström method, 337
Nyström approximation, 449

ObjectRank, 127
online clustering, 66
OOV word, 426
OpenMP, 446
orthogonal matrix, 34
out links, 94
out-degree, 27

PageRank, 111, 312
PageRank complexity, 114
PageRank damping factor, 114
PARAFAC, 44
PARAFAC decomposition, 452
PARPACK, 451
Partial Singular Value Decomposition (PSVD), 414
personalization vector, 113, 121
pivotal, 425
point-clouds, 509
PopRank, 128
positive-definite matrix, 33
power law, 61
power method, 114
powerset, 281
preferential attachment, 62
prestige, 106
Principal Component Analysis (PCA), 97, 438
Probabilistic Latent Semantic Analysis, 329
Probabilistic Latent Semantic Analysis (PLSA), 412, 418
  incremental PLSA, 421
  MAP PLSA, 422
  on-line Probabilistic Latent Semantic Analysis (oPLSA), 424
  PLSA folding-in, 420
  Quasi-Bayes (QB) PLSA, 422
pseudo-relevance feedback, 104
pseudoinverse, 35

quadratic assignment problem, 73
query expansion, 101

random graph generation, 58
random projection, 450
random walk, 59
random walks on hypergraphs, 300
randomized algorithms, 334
  structured random matrix, 337
  subsampled random Fourier transform, 337
range, 32
rank, 32, 110
rank deficient, 32
rank leaks, 113

Rank Removal, 72
rank sinks, 113
rank-r decomposition, 43
RankClus, 129
ranking, 278
RASP algorithm, 449
RDDs, 20
recomendation systems, 59
recommendation, 308
recommender systems, 413
regular, 27
relevance feedback, 104
Resilient Distributed Datasets, 20

SALSA, 118
SCP, 72
Sequential Clique Percolation, 72
similarity measure, 29
SimRank, 119
Simulated Annealing, 68
singular, 33
Singular Value Decomposition, 33
singular value decomposition, 97
Singular Value Decomposition (SVD), 412
singular vectors, 34
sink, 27
small world phenomenon, 61
social image search, 326
soft-thresholding operator, 309
source, 27
Spark, 20
sparsification, 449
spatial localization, 513
SPD, 35
spectral clustering, 412, 433
  Efficient Spectral Clustering on Graphs (ESCG), 450
  Incremental Approximate Spectral Clustering, 437
  incremental spectral clustering, 434
  Landmark-based spectral clustering, 450
spectral hypergraph clustering, 279, 296
spectrum, 515
spider traps, 113
Spouts, 20
standard simplex, 289
statistical topic modeling, 123
status, 110
Storm, 20
Streaming Pattern Discovery in Multiple Time-series (SPIRIT) algorithm, 440
sub-arrays, 40
subgraph, 28, 282
  induced, 28
suffix tree clustering, 133
SVD, 33
svd, 97
symmetric matrix, 28
symmetric normalized Laplacian, 434

teleportation, 113
tensor, 40, 284
  fiber, 40
  inner product, 41
  inner product (over indices), 41
  order, 40
  rank, 43
  rank-one, 43
  simple, 43
  slice, 40
tensor analysis
  Dynamic Tensor Analysis (DTA), 439
  GIGATENSOR, 452
  Incremental Tensor Analysis (ITA), 439
  Independent-Window Tensor Analysis (IWTA), 443
  Moving-Window Tensor Analysis (MWTA), 443
  Offline Tensor Analysis (OTA), 438
  Streaming Tensor Analysis (STA), 439

Window-based Tensor Analysis (WTA), 439
tensors, 412, 438
core tensors, 438
tensor sequences, 438
tensor streams, 438
tensor window, 442
term clustering, 102
term document matrix, 96
term frequency, 96
text-based topic affinity, 121
tf-idf weighting, 97
TKC effect, 118
topic drift, 118
topic sensitivity, 120
topic-sensitive PageRank, 121
total variation, 306
tourism recommendation, 329
tourist place of interest, 329
TrustRank, 124
tube, 40
Tucker decomposition, 45
TwitterRank, 122
two-colorable graphs, 516
typical rank, 44

undirected, 27
undirected links, 312
unfolding, 42
unified neighborhood random walk distance matrix, 456
uniform hypergraph, 283
unnormalized graph Laplacian, 434
usage-based topic affinity, 122

Variational Bayesian Inference (VB), 430
variational inference, 430
vector outer product, 42
vector space model, 99
vertex, 281
vertex access time, 76
vertex degree, 283
vertex incident to a hyperedge, 282

Watts-Strogatz method, 62
weight matrix, 28
weight value, 28
weighted hypergraph, 283