SPRINGER BRIEFS IN COMPUTER SCIENCE

Panagiotis Symeonidis
Andreas Zioupos

Matrix and Tensor Factorization Techniques for Recommender Systems

Springer

SpringerBriefs in Computer Science

More information about this series at http://www.springer.com/series/10028

Panagiotis Symeonidis · Andreas Zioupos

Matrix and Tensor Factorization Techniques for Recommender Systems

Springer

Panagiotis Symeonidis
Faculty of Computer Science
Free University of Bozen-Bolzano
Bozen-Bolzano
Italy

Andreas Zioupos
Thessaloniki
Greece

ISSN 2191-5768 ISSN 2191-5776 (electronic)
SpringerBriefs in Computer Science
ISBN 978-3-319-41356-3 ISBN 978-3-319-41357-0 (eBook)
DOI 10.1007/978-3-319-41357-0

Library of Congress Control Number: 2016943814

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG Switzerland

Contents

Part I
Matrix Factorization Techniques

Chapter 1
Introduction

Abstract Representing data in lower dimensional spaces has been used extensively in many disciplines such as natural language and image processing, data mining, and information retrieval. 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. That is, we can deal with all aforementioned challenges by applying matrix and tensor decomposition methods (also known as factorization methods). In this chapter, we provide some basic definitions and preliminary concepts on dimensionality reduction methods of matrices and tensors. Gradient descent and alternating least squares methods are also discussed. Finally, we present the book outline and the goals of each chapter.

Keywords Matrix decomposition · Tensor decomposition

1.1 Recommender Systems

The Web contains more than 4.9 billion pages until the date this book was published and it is growing rapidly day by day. There is a huge amount of information, which burdens people to find what they may need. To overcome this problem, we often rely on recommendations from others who have more experience on a topic. In the Web, this is attained with the help of a collaborative filtering (CF) algorithm, which provides recommendations based on the suggestions of users, who have similar preferences for products. Basically, it is an algorithm for matching people with similar interests under the assumption that similar people like similar products. These kind of recommendations are provided from systems, known as recommender systems. Recommender systems use techniques, which are widely studied in research communities of information retrieval, machine learning, and data mining [16, 36]. These systems have been developed to effectively support the customer's decision-making process mainly for commercial purposes (i.e., what product to buy on Amazon.com, what TV show or movie to rent on Netflix.com, etc.).

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_1

In the following, we will discuss an example of a simple recommender system with four users, who have rated four movies, as shown in Fig. 1.1.

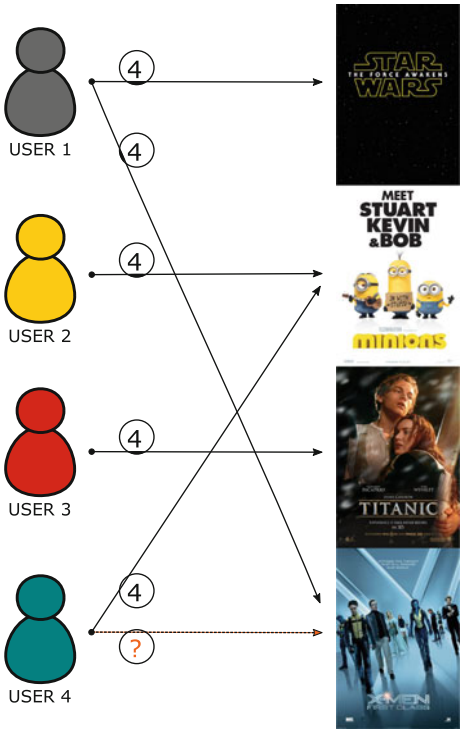

Fig. 1.1 Four users have rated four movies using a [1–5] rating scale

Figure 1.1 presents a weighted user-movie bipartite graph, where each edge between a user and a movie has a weight which indicates the degree of preference of a user for that movie (using a [0–5] rating scale). For reasons of clarity, in Fig. 1.1, we present only "positive" ratings ($\geq$4). Let us assume in our running example that we want to predict the rating of user 4 on movie 4 (X-Men). To show this assumption, we use a dotted line for the edge that connects user 4 with movie 4 and the possible rating is shown with a question mark. The challenge of a recommender system is to correctly predict a rating for those items for which a user has not expressed explicitly her preference (with no rating value at all). In case our recommender system predicts this rating as positive (i.e., higher than 4 in the rating scale 0–5), then, it could recommend movie 4 (X-Men) to the target user 4.

Our example of Fig. 1.1 can be represented with a user–item rating matrix A, which is shown in Fig. 1.2a. Please notice that in contrast to Fig. 1.1, Fig. 1.2 also presents ratings with values 0, 1, 2, and 3. In the case that we read matrix A horizontally, Fig. 1.2 represents the ratings that a user U_i gives to a set of movies I_j, where

$j \geq 1$. In the case that we read the matrix A vertically, Fig. 1.2 represents the ratings that one movie (i.e., I_1) receives from several users U_i, where $i \geq 1$.

To ease the discussion, we will use the running example illustrated in Fig. 1.2 where I_{1-4} are items and U_{1-4} are users. As shown, the example data set is divided into training and test sets. The null cells (no rating) are presented as zeros.

(a)

	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

(b)

	I_1	I_2	I_3	I_4
U_4	1	4	1	?

Fig. 1.2 **a** Training set (3 × 4), **b** Test set (1 × 4)

1.2 Recommender Systems in Social Media

Online social networks (OSNs) contain gigabytes of data that can be mined to provide product recommendations to a target user. Besides explicit friendship relations among users, there are also other implicit relations. For example, users can co-comment on products and can co-rate them. Thus, item recommendation can be provided in such systems based on the suggestions of our friends whom we trust. Recommender systems are widely used by OSNs to stimulate users to extend their personal social graph or for marketing purposes by recommending products that their friends have liked. Moreover, recommender systems can act like filters, trying to provide the right personalized information to each different user. Typically, a recommendation algorithm takes as input the preferences of the user and her friends from their profile and conscripts them to yield recommendations for new ties such as friends, companies, places, products, etc.

In the following, we will discuss a more extended example, where we will try to provide recommendations to user 4 by exploiting both her ratings on movies and her friendship network, as shown in Fig. 1.3, which consists of two layers. The first layer contains the friendship network among users, whereas the second layer shows the movies, which are rated only positively by users.

Our recommender system's task is to predict a rating for user 4 on movie 4. As shown, user 4 is a friend of user 2, who does not like the X-Men movie at all (rated it with 0 in the [0–5] rating scale as shown in Fig. 1.2a). Based on the theory of homophily, which claims that friends may share common characteristics (i.e., beliefs, values, preferences, etc.), we cannot predict a high rating for user 4 on the X-Men movie. This is a simple example of how a social network (i.e., friendship network) can help a recommender system to leverage the quality of its recommendations by exploiting the information from the friendship network.

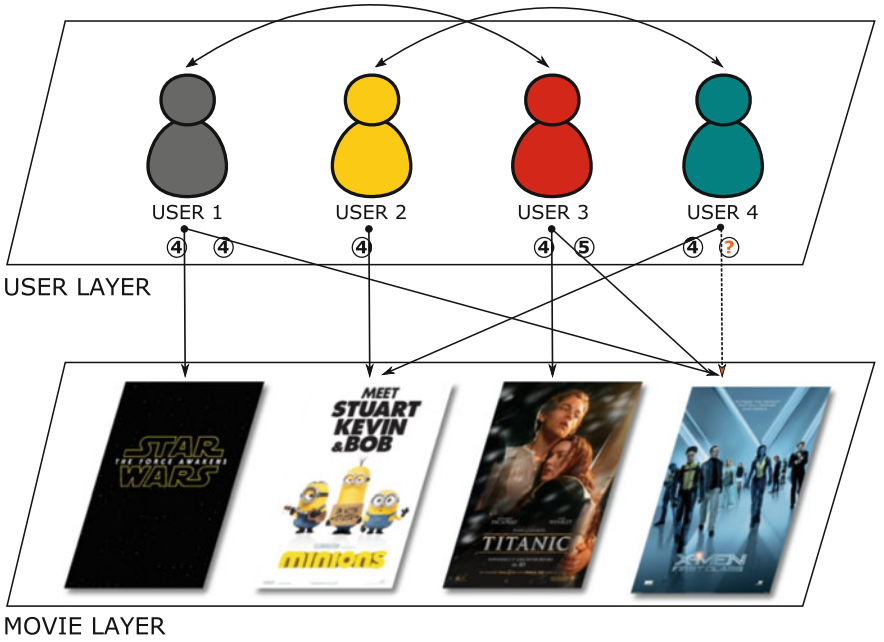

Fig. 1.3 The users' friendship network and the user-movie rating information

1.3 Matrix Factorization

Recommender systems mainly base their suggestions on rating data of two entities (users and items), which are often placed in a matrix with one representing users and the other representing items of interest. For example, Netflix collects ratings for movies using the five-star selection schema, and TiVo users indicate their preferences for TV shows by pressing thumbs-up and thumbs-down buttons. These ratings are given explicitly by users creating a sparse user–item rating matrix, because an individual user is likely to rate only a small fraction of the items that belong to the item set. Another challenging issue with this user–item rating matrix is scalability of data (i.e., the large number of possible registered users or inserted items), which may affect the time performance of a recommendation algorithm.

We can deal with all aforementioned challenges by applying matrix decomposition methods (also known as factorization methods). *Matrix factorization* denotes a process, where a matrix is factorized into a product of matrices. A matrix factorization method is useful for solving plenty of problems, both analytical and numerical; an example of a numerical problem is the solution of linear equations and eigenvalue problems. Its importance relies on the exploitation of latent associations that exist

in the data among participating entities (e.g., between users and items). In a trivial form, the matrix factorization method uses two matrices, which hold the information of correlation between the user-feature and item-feature factors, respectively.

Figure 1.4 shows an example of latent factors, which could be revealed after performing matrix decomposition. As shown, the X′X axis divides both people and movies according to sex (e.g., male or female). When a movie is closer to the female part of X′X axis, it means that this movie is most popular among women rather than

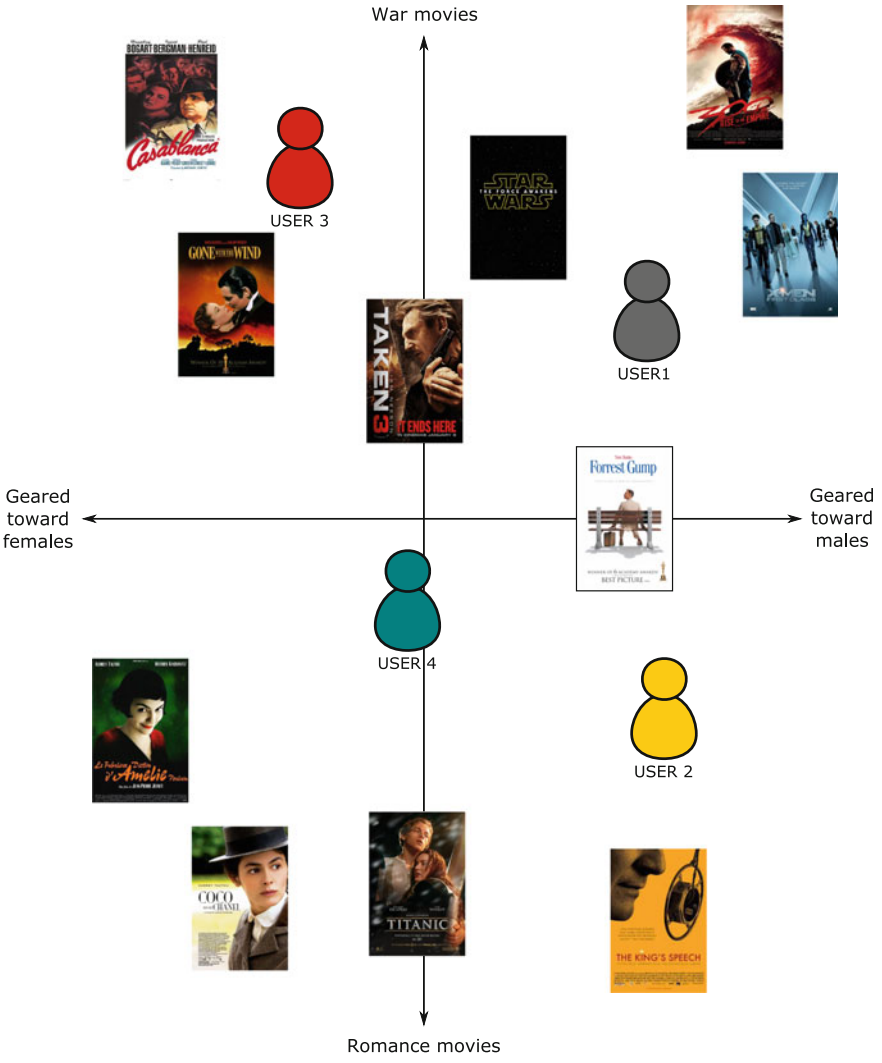

Fig. 1.4 A simplified illustration of the latent factors, which characterizes both users and movies using two axes—male versus female and war-like versus romantic

men. The Y′Y axis divides people and movies as "war-like" and "romantic." A "war-like" viewer is assumed to prefer movies showing blood and deaths. In contrast, a "romantic" viewer chooses movies that present love and passion. To predict a user's rating of a movie, we can compute the dot product of the movie's and user's [x, y] coordinates on the graph. In addition, Fig. 1.4 shows where movies and users might fall on the basic two dimensions. For example, we would expect *user 3* to love "*Casablanca*," to hate "*The King's Speech*," and to rate "*Amelie*" above average. Note that some movies (i.e., "*Taken 3*") and users (i.e., *user 4*) would be characterized as fairly neutral on these two dimensions.

One strong point of matrix factorization is that it also allows the incorporation of additional information. When explicit feedback is not available, recommender systems can infer user preferences using implicit feedback, which indirectly reflects opinions by observing user behavior including purchase history, browsing history, search patterns, or even mouse movements. Implicit feedback usually denotes the presence or absence of an event, so it is typically represented by a densely filled matrix [26].

There are many matrix factorization methods (see Chap. 2). The popularity of these methods and the motivation to study them more in recent years came by their advantage of combining high-performance scalability with good predictive accuracy.

1.4 Tensor Factorization

Standard CF-based algorithms operate on matrices (second-order tensors) representing relations between users and items. However, real-world problems usually consist of more than two participating entities. For example, social tagging systems (STSs) mainly consist of three participating entities (users, items, and tags). Moreover, in location-based social networks (LBSNs), we have also three interacting entities (users, locations, and tags). In LBSNs, users can share location-related information with each other to leverage the collaborative social knowledge. LBSNs consist of a new social structure made up of individuals connected by interdependency derived from their locations in the physical world as well as their location-tagged media content, such as photographs, videos, and texts. As shown in Fig. 1.5, users visit locations in the real world and provide geo-tagged information content (e.g., comments, photographs, videos). In particular, Fig. 1.5 presents three layers, namely user, location, and content. It is obvious that someone could exploit information from each layer independently to leverage recommendations. However, in a more advanced case, we could also exploit ternary relation among entities (i.e., user, location, and content), which goes through all layers.

Because of the ternary relation of data in many cases (e.g., STSs, LBSNs, etc.), many recommendation algorithms originally designed to operate on matrices cannot be applied. Higher order problems put forward new challenges and opportunities for recommender systems. For example, ternary relation of STSs can be represented as a third-order tensor $\mathcal{A} = (a_{u,i,t}) \in \mathbb{R}^{|U|\times|I|\times|T|}$. Symeonidis et al. [44],

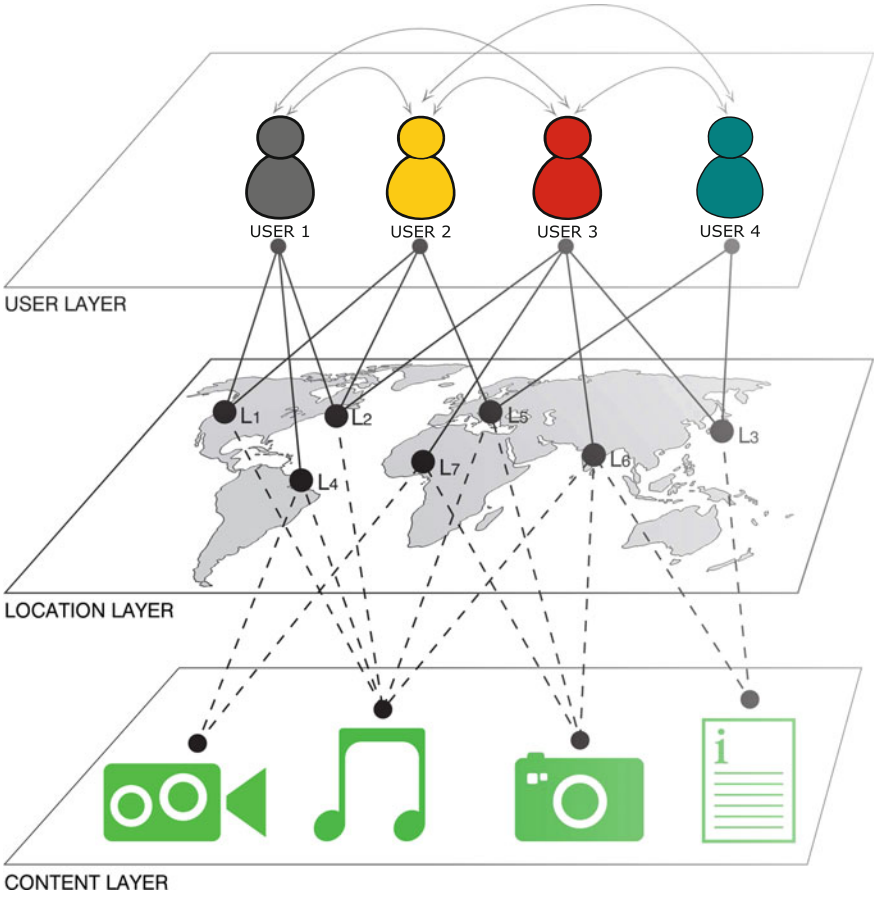

Fig. 1.5 Visual representation of users, locations, and content (i.e., photographs/videos, tags, etc.)

for example, proposed to interpret the user assignment of a tag on an item, as a binary tensor where 1 indicates observed tag assignments and 0 missing values (see Fig. 1.6):

$$a_{u,i,t} := \begin{cases} 1, & \text{if user } u \text{ assigns on item } i \text{ tag } t \\ 0, & \text{otherwise} \end{cases}$$

Tensor factorization techniques can be employed in order to exploit the underlying latent semantic structure in tensor $\mathcal{A}$. While the idea of computing low-rank tensor approximations has already been used in many disciplines such as natural language, image processing, data mining and information retrieval [11, 13, 24, 28, 42, 43, 46], just a few years ago, it was applied to recommendation problems in STSs and

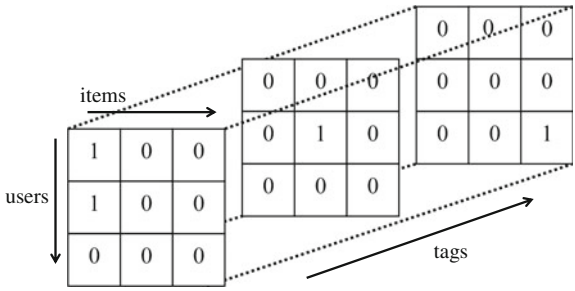

Fig. 1.6 Tensor representation of a STS where positive feedback is interpreted as 1 (i.e., $a_{utr} := 1$) and the rest as 0 (i.e., $a_{utr} := 0$)

LBSNs. The basic idea is to transform the recommendation problem as a third-order tensor completion problem, by trying to predict nonobserved entries in $\mathcal{A}$.

1.5 Mathematical Background and Notation

In this section, we provide all important notations of variables and symbols, which will be used throughout the book. Moreover, we provide some basic theorems or mathematical definitions as preliminary knowledge to help the reader understand more easily the concepts that will be discussed later.

The notation of every matrix, variable symbol, and any other basic mathematical term is presented in Tables 1.1 and 1.2, respectively.

Linear algebra plays an important role in matrix and tensor decomposition. Therefore, preliminary concepts on matrices drawn from linear algebra are reviewed in this section.

A *diagonal matrix* (also known as *square matrix*) is a matrix in which entries outside the main diagonal are all zero. Thus, a matrix A with $N \times N$ dimensions is diagonal if the following constraint is satisfied:

$$a_{ij} = 0 \quad if \quad i \neq j \quad \forall i, j \in \{1, 2, \ldots, N\} \tag{1.1}$$

The (column) *rank* of a matrix $A \in \mathbb{R}^{N \times M}$ is defined to be the number of linearly independent column vectors. The (row) *rank* of A is defined to be the number of linearly independent row vectors of A.

A square matrix $A \in \mathbb{R}^{N \times N}$ is called *invertible* (or *nonsingular*), if there is a matrix $B \in \mathbb{R}^{N \times N}$ such that:

$$AB = I \quad \text{and} \quad BA = I \tag{1.2}$$

where $I \in \mathbb{R}^{N \times N}$ is the *identity matrix*. A square matrix that is not invertible is called *singular*. A is singular, if its rank is less than N.

Table 1.1 Definition of matrices and variables that will be used throughout this book

Matrices and variables		
I	Identity matrix	An $n \times n$ square matrix with 1 s on the main diagonal and 0 s elsewhere
$\mathbb{R}$	Real numbers	The set of real numbers
A	User–item rating matrix	This is the user–item rating matrix, which holds ratings of users on movies. Cells with zeros denote the absence of a rating
$\hat{A}$	Prediction matrix	This is a user–item rating matrix, which holds predicted ratings
U	User-latent feature matrix	This matrix holds preferences of users over items on a latent feature space of dimensionality f. In the CUR method, U has another meaning (see Sect. 2.6)
V	Item-latent feature matrix	This matrix expresses how much an item is preferred by users on a latent feature space of dimensionality f
Λ	Eigenvalue matrix	An $r \times r$ diagonal matrix filled with nonzero eigenvalues of matrix A
E	Eigenvector matrix	E ($n \times r$ matrix) stores eigenvectors of A
Γ		Symmetric nonnegative matrix with diagonal elements equal to zero and other elements greater than zero
C		Randomly chosen set of r columns of matrix A
R		Randomly chosen set of r rows of matrix A
F	Friendship matrix	Stores the friendship information among users. 1s declare friendship and 0s no friendship
λ	Eigenvalue	An eigenvalue λ
$\mathbf{e}$	Eigenvector	An eigenvector e
η	eta	This variable controls the size of the step toward minimization of an objective function
β	beta	The coefficient that regularize predicted users' ratings on items
γ	gamma	The coefficient that regulates the contribution of the friendship network

An *eigenvector* of a square matrix $A^{N \times N}$ is a nonzero vector $\mathbf{e} \in \mathbb{R}^N$ that satisfies the following equation:

$$A\mathbf{e} = \lambda\mathbf{e}, \tag{1.3}$$

meaning that the vector $A\mathbf{e}$ follows the direction of $\mathbf{e}$. λ is the *eigenvalue* of A corresponding to the eigenvector $\mathbf{e}$.

Table 1.2 Definition of mathematical symbols that will be used throughout this book

Symbols		
i, j	Indices	Denote the position of an element inside a matrix
$\neq$	Not equal	Is not equal to
$\leq$	Inequality	Is less than or equal to
$\geq$	Inequality	Is greater than or equal to
$\approx$	Approximately equal	Is approximately equal to
$B^\top$	Transpose symbol	Reflects matrix B over its main diagonal (which runs from top-left to bottom-right) to obtain $B^\top$
B^{-1}	Inverse symbol	The inverse of a square matrix is a matrix such that $BB^{-1} = I$
$\forall$	For all	
$\in$	Set membership	Is an element of
$\notin$	Set membership	Is not an element of
$\rightarrow$	Material implication	if . . . then
$\Longleftrightarrow$	Material equivalence	if and only if
$\mathbf{b}$	Vector	Euclidean vector
$\wedge$	And	
$\vee$	Or	
$\cdot$	Dot product	The dot product of vectors or matrices
$\| \ldots \|$	Norm	Euclidean norm
$\sum$	Sum	The sum of elements from beginning to end
∂	Partial derivative	The partial derivative of a function

A square matrix $A \in \mathbb{R}^{N \times N}$ is called *orthogonal*, if column vectors of A form an orthonormal set in $\in \mathbb{R}^N$. In other words, an *orthogonal matrix* is a square matrix with real numbers as entries, whose columns and rows are orthogonal unit vectors as shown below:

$$AA^\top = A^\top A = I \quad : I \text{ is the identity matrix} \tag{1.4}$$

This leads to equivalent characterization: a matrix A is orthogonal if its transpose is equal to its inverse:

$$A^\top = A^{-1} \tag{1.5}$$

The *Frobenius norm* $||A||$ of a matrix is given by:

$$||A|| = \sum_{i=1}^{N} \sum_{j=1}^{M} {a_{ij}}^2. \tag{1.6}$$

1.6 Book Outline

In the sequel, a brief introduction to each chapter of the book follows:

Chapter 2. **Related Work on Matrix Factorization**
In this chapter, we provide the related work on basic matrix decomposition methods. The first method that we discuss is known as eigenvalue decomposition, which decomposes the initial matrix into a canonical form [1, 20, 34, 35, 48]. The second method is nonnegative matrix factorization (NMF), which factorizes the initial matrix into two smaller matrices with the constraint that each element of the factorized matrices should be nonnegative [3, 7, 8, 14, 18, 22, 29–31, 47]. The third method is probabilistic matrix factorization (PMF), which scales well to large data sets. The PMF method performs well on very sparse and imbalanced data sets using spherical Gaussian priors. The last but one method is probabilistic latent semantic analysis (PLSA) which is based on a mixture decomposition derived from a latent class model. This results in a more principled approach which has a solid foundation in statistics. The last method is CUR decomposition, which confronts the problem of density in factorized matrices (a problem that is faced when handling the SVD method) [15, 32, 33].

Chapter 3. **Performing SVD on Matrices and Its Extensions**
In this chapter, we describe singular value decomposition (SVD), which is applied on recommender systems [2, 5, 9, 12, 19, 27, 40, 41]. We discuss in detail the mathematical background and present (step by step) the SVD method using a toy example of a recommender system. We also describe UV decomposition [38] in detail, which is an instance of SVD, as we have mathematically proven. We minimize an objective function, which captures the error between the predicted and the real value of a user's rating. We also provide a step-by-step implementation of UV decomposition using a toy example, which is followed by a short representation of the algorithm in pseudocode form [4, 17, 49]. Finally, an additional constraint of friendship is added to the objective function to leverage the quality of recommendations [25].

Chapter 4. **Experimental Evaluation on Matrix Decomposition Methods**
In this chapter, we study the performance of described SVD and UV decomposition algorithms, against an improved version of the original item-based CF algorithm [23, 39] combined with SVD. For the UV decomposition method, we will present the appropriate tuning of parameters of its objective function to have an idea of how we can get optimized values of its parameters. We will also answer the question if these values are generally accepted or if they should be different for each data set. The metrics we will use are root-mean-square error (RMSE), precision, and recall. The size of a training set is fixed at 75 %, and we perform a fourfold cross-validation.

Chapter 5. **Related Work on Tensor Factorization**
In this chapter, we provide a preliminary knowledge overview of tensors. Moreover, we provide the related work on tensor decomposition methods. The first method that is discussed is the Tucker Decomposition (TD) method [45], which is the underlying tensor factorization model of Higher Order Singular Value Decomposition [28]. TD decomposes a tensor into a set of matrices and one small core tensor. The second one is the PARAFAC method (PARAllel FACtor analysis) [10, 21], which is the same as the TD method with the restriction that the core tensor should be diagonal. The third one is the Pairwise Interaction Tensor Factorization method [37], which is a special case of the TD method with linear runtime both for learning and prediction. The last method that is analyzed is the low-order tensor decomposition (LOTD). This method has low functional complexity, is uniquely capable of enhancing statistics, and avoids overfitting compared with traditional tensor decompositions such as TD and PARAFAC [6].

Chapter 6. **Performing HOSVD on Tensors and Its Extensions**
In this chapter, we describe tensor decomposition for recommender systems in detail. We will use—as a toy example—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 will also discuss other problems in tensor decomposition, e.g., actualization and scalability.

Chapter 7. **Experimental Evaluation on Tensor Decomposition Methods**
In this chapter, we will provide experimental results of tensor decomposition methods on real data sets in STSs. We will discuss the criteria that we will set for testing all algorithms and the experimental protocol we will follow. Moreover, we will discuss the metrics that we will use (i.e., Precision, Recall, root-mean-square error, etc.). Our goal is to present the main factors that influence the effectiveness of algorithms.

Chapter 8. **Conclusions and Future Work**
In this chapter, we will discuss the main conclusions of the experimental evaluation, limitations of each algorithm, and will provide future research directions.

References

1. Bensmail, H., Celeux, G.: Regularized gaussian discriminant analysis through eigenvalue decomposition. J. Am. Stat. Assoc. **91**(436), 1743–1748 (1996)

2. Berry, M., Dumais, S., O'Brien, G.: Using linear algebra for intelligent information retrieval. SIAM Rev. **37**(4), 573–595 (1994)
3. Berry, M.W., Browne, M., Langville, A.N., Paul Pauca, V., Plemmons, R.J.: Algorithms and applications for approximate nonnegative matrix factorization. Comput. Stat. Data Anal. **52**(1), 155–173 (2007)
4. Bhargava, P., Phan, T., Zhou, J., Lee, J.: Who, what, when, and where: multi-dimensional collaborative recommendations using tensor factorization on sparse user-generated data. In: Proceedings of the 24th International Conference on World Wide Web, WWW'15. Republic and Canton of Geneva, Switzerland, pp. 130–140. International World Wide Web Conferences Steering Committee (2015)
5. Brand, M.: Incremental singular value decomposition of uncertain data with missing values. In: Proceedings of the 7th European Conference on Computer Vision (ECCV), pp. 707–720. Copenhagen, Denmark (2002)
6. Cai, Y., Zhang, M., Luo, D., Ding, C., Chakravarthy, S.: Low-order tensor decompositions for social tagging recommendation. In: Proceedings of the Fourth ACM International Conference on Web Search and Data Mining, WSDM'11, New York, NY, USA, pp. 695–704. ACM (2011)
7. Campbell, S.L., Poole, G.D.: Computing nonnegative rank factorizations. Linear Algebra Appl. **35**, 175–182 (1981)
8. Cao, B., Shen, D., Sun, J.-T., Wang, X., Yang, Q., Chen, Z.: Detect and track latent factors with online nonnegative matrix factorization. In: IJCAI, pp. 2689–2694 (2007)
9. Cao, L.: Singular value decomposition applied to digital image processing. Division of Computing Studies Arizona State University Polytechnic Campus, Mesa, Arizona, 85212 (2006)
10. Carroll, J.D., Chang, J.J.: Analysis of individual differences in multidimensional scaling via an n-way generalization of eckart-young decomposition. Psychometrika **35**, 283–319 (1970)
11. Chen, S., Wang, F., Zhang, C.: Simultaneous heterogeneous data clustering based on higher order relationships. In: Proceedings of the Seventh IEEE International Conference on Data Mining Workshops, ICDMW'07, pp. 387–392, Washington, DC, USA. IEEE Computer Society (2007)
12. 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)
13. Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., Harshman, R.: Indexing by latent semantic analysis. J. Am. Soc. Inf. Sci. **41**(6), 391–407 (1990)
14. Dhillon, I.S., Sra, S.: Generalized nonnegative matrix approximations with bregman divergences. In: NIPS, pp. 283–290 (2005)
15. Drineas, P., Kannan, R., Mahoney, M.W.: Fast monte carlo algorithms for matrices III: computing a compressed approximate matrix decomposition. SIAM J. Comput. **36**(1), 184–206 (2006)
16. Fodor, I.: A survey of dimension reduction techniques. Technical Report (2002)
17. Forsati, R., Mahdavi, M., Shamsfard, M., Sarwat, M.: Matrix factorization with explicit trust and distrust side information for improved social recommendation. ACM Trans. Inf. Syst. **32**(4), 17:1–17:38 (2014)
18. Fulekar, M.H.E.: Bioinformatics: Applications In Life And Environmental Sciences. Daya Publishing House (2009)
19. 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)
20. Golub, G.H., Van Loan, C.F.: Matrix Computations, 3rd edn. Johns Hopkins University Press, Baltimore, MD, USA (1996)
21. Harshman, R.A.: Foundations of the parafac procedure: models and conditions for an 'exploratory' multimodal factor analysis. In: UCLA Working Papers in Phonetics, pp. 1–84 (1970)
22. Kalofolias, V., Gallopoulos, E.: Computing symmetric nonnegative rank factorizations. Linear Algebra Appl. **436**(2), 421–435 (2012). Special Issue devoted to the Applied Linear Algebra Conference (Novi Sad 2010)

23. Karypis, G.: Evaluation of item-based top-n recommendation algorithms. In: Proceedings of ACM CIKM Conference, pp. 247–254 (2001)
24. 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
25. Koren, Y.: Factorization meets the neighborhood: a multifaceted collaborative filtering model. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'08, New York, NY, USA, pp. 426–434. ACM (2008)
26. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8), 30–37 (2009)
27. de Lathauwer, L., de Moor, B., Vandewalle, J.: A multilinear singular value decomposition. SIAM J. Matrix Anal. Appl. **21**(4), 1253–1278 (2000)
28. De Lathauwer, L., De Moor, B., Vandewalle, J.: A multilinear singular value decomposition. SIAM J. Matrix Anal. Appl. **21**(4), 1253–1278 (2000)
29. Lee, D.D., Seung, H.S.: Learning the parts of objects by nonnegative matrix factorization. Nature **401**, 788–791 (1999)
30. Lee, D.D., Seung, H.S.: Algorithms for non-negative matrix factorization. In: NIPS, pp. 556–562 (2000)
31. Lin, C.-J.: On the convergence of multiplicative update algorithms for nonnegative matrix factorization. IEEE Trans. Neural Netw. **18**(6), 1589–1596 (2007)
32. Mahoney, M.W., Drineas, P.: Cur matrix decompositions for improved data analysis. Proc. Natl. Acad. Sci. **106**(3), 697–702 (2009)
33. Mahoney, M.W., Maggioni, M., Drineas, P.: Tensor-cur decompositions for tensor-based data. SIAM J. Matrix Anal. Appl. **30**(3), 957–987 (2008)
34. Moler, C.B., Stewart, G.W.: An algorithm for generalized matrix eigenvalue problems. SIAM J. Numer. Anal. **10**(2), 241–256 (1973)
35. Quarteroni, A., Sacco, R., Saleri, F.: Numerical Mathematics. Texts in Applied Mathematics. Springer (2000)
36. Rajaraman, A., Ullman, J.D.: Mining of Massive Datasets. Cambridge University Press, New York, NY, USA (2011)
37. Rendle, S.: Pairwise interaction tensor factorization for personalized tag recommendation
38. Roweis, S.T., Saul, L.K.: Nonlinear dimensionality reduction by locally linear embedding. Science **290**(5500), 2323–2326 (2000)
39. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item-based collaborative filtering recommendation algorithms. In: Proceedinmgs of WWW Conference, pp. 285–295 (2001)
40. 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)
41. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Incremental singular value decomposition algorithms for highly scalable recommender systems. In: Proceedings of the 5th International Conference in Computers and Information Technology (2002)
42. Shashua, A., Hazan, T.: Non-negative tensor factorization with applications to statistics and computer vision. In: ICML'05: Proceedings of the 22nd International Conference on Machine Learning, pp. 792–799. ACM (2005)
43. Sun, J., Shen, D., Zeng, H., Yang, Q., Lu, Y., Chen, Z.: Cubesvd: a novel approach to personalized web search. In: World Wide Web Conference, pp. 382–390 (2005)
44. Symeonidis, P., Nanopoulos, A., Manolopoulos, Y.: Tag recommendations based on tensor dimensionality reduction. In: RecSys'08: Proceedings of the 2008 ACM Conference on Recommender Systems, pp. 43–50. ACM (2008)
45. Tucker, L.: Some mathematical notes on three-mode factor analysis. Psychometrika 279–311 (1966)
46. Wang, H., Ahuja, N.: A tensor approximation approach to dimensionality reduction. Int. J. Comput. Vis. 217–229 (2007)

47. X, W., Gao, X.: Non-negative matrix factorization by maximizing correntropy for cancer clustering. BMC Bioinform. **14**, 107 (2013)
48. Weisstein, E.W.: Eigen decomposition from mathworld.wolfram.com. http://mathworld.wolfram.com/EigenDecomposition.html
49. Yuan, Q., Chen, L., Zhao, S.: Factorization vs. regularization: fusing heterogeneous social relationships in top-n recommendation (2011)

Chapter 2
Related Work on Matrix Factorization

Abstract In this chapter, we provide the related work of basic matrix decomposition methods. The first method that we discuss is known as eigenvalue decomposition, which decomposes the initial matrix into a canonical form. The second method is nonnegative matrix factorization (NMF), which factorizes the initial matrix into two smaller matrices with the constraint that each element of the factorized matrices should be nonnegative. The third method is latent semantic indexing (LSI), which applies singular value decomposition (SVD) that uses singular values of the initial matrix to factorize it. The last method is CUR decomposition, which confronts the problem of high density in factorized matrices (a problem that is faced when using the SVD method). This chapter concludes with a description of other state-of-the-art matrix decomposition techniques.

Keywords Matrix decomposition

2.1 Dimensionality Reduction on Matrices

Many problems in our life are represented with matrices. Due to high dimensionality of data in these problems, the initial matrix is usually factorized into two or more "smaller" matrices. These matrices have the advantage of smaller dimensions, resulting in reduced required storage space and less required time for processing them. Therefore, they can be processed more efficiently by algorithms than the initial matrix. There are many methods [19–21] on how to decompose a matrix and deal with a high-dimensional data set.

Principal component analysis (PCA) is a data mining technique that replaces the high-dimensional original data by its projection onto the most important axes. This technique maps linearly the data to a lower dimensional matrix in such a way that the

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_2

variance of data in the low-dimensional representation is maximized. It is a simple method, which is based on eigenvalues and the eigenvectors of a matrix, which will be discussed in the next section.

2.2 Eigenvalue Decomposition

In this section, we present a decomposition method known as *eigenvalue decomposition*. In linear algebra, the eigenvalue decomposition method is the factorization of a matrix A into a canonical form. The eigenvalues and the eigenvectors of A are used to represent the matrix. To apply this method on a matrix, the matrix should not only be square but also diagonalizable [1, 11, 18].

Let A be a square and diagonalizable matrix, and let λ be a constant and $\mathbf{e}$ a column nonzero vector with the same number of rows as A. Then λ is an eigenvalue of A and $\mathbf{e}$ is the corresponding eigenvector of A if:

$$A\mathbf{e} = \lambda\mathbf{e} \tag{2.1}$$

For a matrix A of rank r, we can group r nonzero eigenvalues in an $r \times r$ diagonal matrix Λ and their eigenvectors in an $n \times r$ matrix E. So we have:

$$AE = E\Lambda \tag{2.2}$$

Moreover, in case that the rank r of the matrix A is equal to its dimension n, then the matrix A can be factorized as:

$$A = E\Lambda E^{-1} \tag{2.3}$$

which is a diagonalization similar to SVD which will be described particularly in the next chapter.

Next, based on Eq. 2.3, we perform eigenvalue decomposition on the following 2×2 matrix A:

$$A = \begin{bmatrix} 1 & 0 \\ 1 & 3 \end{bmatrix} \tag{2.4}$$

The task is to decompose matrix A into a diagonal matrix through multiplication of a nonsingular matrix B:

$$B = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \in \mathbb{R}^{2\times 2} \tag{2.5}$$

Considering the theory of eigenvalues and eigenvectors of a matrix and Eq. 2.3, we take:

$$
\begin{gathered}
A = B\Lambda B^{-1} \iff \\
AB = B\Lambda \iff \\
\begin{bmatrix} 1 & 0 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}
\end{gathered} \tag{2.6}
$$

Using linear algebra rules, the above equation can be written as:

$$
\begin{gathered}
\left.\begin{gathered} \begin{bmatrix} 1 & 0 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} a \\ c \end{bmatrix} = \begin{bmatrix} a\lambda_1 \\ c\lambda_1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} b \\ d \end{bmatrix} = \begin{bmatrix} b\lambda_2 \\ d\lambda_2 \end{bmatrix} \end{gathered}\right\} \\
\iff \\
\left.\begin{gathered} \begin{bmatrix} 1 & 0 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} a \\ c \end{bmatrix} = \lambda_1 \begin{bmatrix} a \\ c \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} b \\ d \end{bmatrix} = \lambda_2 \begin{bmatrix} b \\ d \end{bmatrix} \end{gathered}\right\}
\end{gathered} \tag{2.7}
$$

The next step is to let $\mathbf{a} = \begin{bmatrix} a \\ c \end{bmatrix}$ and $\mathbf{b} = \begin{bmatrix} b \\ d \end{bmatrix}$, this gives us two vector equations:

$$
\left.\begin{gathered} A\mathbf{a} = \lambda_1 \mathbf{a} \\ A\mathbf{b} = \lambda_2 \mathbf{b} \end{gathered}\right\} \tag{2.8}
$$

Equation 2.8 can be described with a single vector equation involving two solutions as eigenvalues. Based on Eq. 2.1, with λ representing the two eigenvalues λ_1 and λ_2, respectively, $\mathbf{e}$ is $\mathbf{a}$ and $\mathbf{b}$. Now shifting the $\lambda\mathbf{e}$ to the left-hand side of the equation we get:

$$
\begin{gathered}
A\mathbf{e} - \lambda\mathbf{e} = 0 \iff \\
(A - \lambda I)\mathbf{e} = 0 \iff \\
\xrightarrow{B\text{:}nonsingular \iff \mathbf{e}\text{:}nonzero} \\
(A - \lambda I) = 0 \iff \\
\begin{vmatrix} 1-\lambda & 0 \\ 1 & 3-\lambda \end{vmatrix} = 0 \iff \\
(1-\lambda)(3-\lambda) = 0
\end{gathered} \tag{2.9}
$$

So we can now compute solutions and take eigenvalues of the matrix A which are $\lambda_1 = 1$ and $\lambda_2 = 3$. The resulting matrix from the *eigenvalue decomposition* of A is:

$$
\lambda = \begin{bmatrix} 1 & 0 \\ 0 & 3 \end{bmatrix}
$$

Putting solutions back into Eq. 2.7, we get:

$$\left.\begin{bmatrix}1 & 0\\ 1 & 3\end{bmatrix}\begin{bmatrix}a\\ c\end{bmatrix} = 1\begin{bmatrix}a\\ c\end{bmatrix}\right\}$$
$$\left.\begin{bmatrix}1 & 0\\ 1 & 3\end{bmatrix}\begin{bmatrix}b\\ d\end{bmatrix} = 3\begin{bmatrix}b\\ d\end{bmatrix}\right\} \tag{2.10}$$

Solving equations we have:

$$a = -2c, \quad a \in \mathbb{R} \tag{2.11}$$
$$\wedge$$
$$b = 0, \quad d \in \mathbb{R} \tag{2.12}$$

Finally, matrix B is $\begin{bmatrix}-2c & 0\\ c & d\end{bmatrix}$ and the solution to the requested example is:

$$\begin{bmatrix}1 & 0\\ 1 & 3\end{bmatrix} = \begin{bmatrix}-2c & 0\\ c & d\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & 3\end{bmatrix}\begin{bmatrix}-2c & 0\\ c & d\end{bmatrix}^{-1}, \quad c, d \in \mathbb{R}$$

2.3 Nonnegative Matrix Factorization

Another widely known method in dimensionality reduction and data analysis is *nonnegative matrix factorization* (NMF). In this section, we will discuss how the NMF algorithm works and apply it to the training data of the running example.

The NMF algorithm factorizes a matrix A in two matrices U and V, with the property that all three matrices have no negative elements. This nonnegativity makes resulting matrices more suitable for the clustering of objects.

Let us assume that $\mathbf{a}_i, \ldots, \mathbf{a}_M$ are M nonnegative initial vectors and we organize them as the columns of a nonnegative data matrix A. NMF produces a small set of K nonnegative representative vectors $\mathbf{u}_i, \ldots, \mathbf{u}_K$ that can be multiplied with $\mathbf{v}_i, \ldots, \mathbf{v}_K$ vectors to approximate initial vectors $\mathbf{a}_i, \ldots, \mathbf{a}_M$ [14], as follows:

$$A \approx UV \tag{2.13}$$

and

$$\mathbf{a}_i \approx \sum_{k=1}^{K} u_{mk} v_{kn}, \quad 1 \leq m \leq M, \quad 1 \leq n \leq N \tag{2.14}$$

Please notice that multiplied u_{nk} and v_{km} coefficients are restricted to being nonnegative [6]. There are several ways in which matrices U and V of Eq. 2.13 can be computed. Lee and Seung [13] proposed the definition of a cost function (i.e., $\|A - UV\|^2$), which can be minimized using either multiplicative update rules or additive update rules of the gradient descent method. In other words, they use a cost function for measuring the divergence between A and UV. The cost function quantifies the approximation error in (2.13), based on the Frobenius norm (square of the *Euclidean distance*) between matrices A and UV:

$$\|A - UV\|^2 = \sum_{k=1}^{K} (a_{mn} - u_{mk} v_{kn})^2 \tag{2.15}$$

is lower bounded by zero and clearly vanishes if and only if $A = UV$.

Thus, an NMF variant arises from the following minimization problem:

$$\textbf{Minimize } \|A - UV\|^2, \textbf{ subject to } U, V \geq 0.$$

Please notice that the Frobenius norm cost function is not convex in U and V simultaneously. The gradient descent can be applied, but it has slow convergence and is sensitive to the choice of gradience step size [31].

The *multiplicative update rules* can be used for iteratively solving the abovementioned minimization problem. They take the following form for the Frobenius norm:

$$U \leftarrow U \frac{AV^T}{AVV^T}, V \leftarrow V \frac{U^T A}{U^T UV} \tag{2.16}$$

Moreover, Dhillon and Sra [5] proposed multiplicative update rules that incorporate weights for the importance of elements of the approximation matrix. As already mentioned, the objective function $\|A - UV\|^2$ is convex in U only or V only. However, it is not convex in both variables together. Thus, we can only guarantee for finding a local minimum solution, rather than a global minimum of the cost function. Moreover, since the problem does not have an exact solution, in general, the computation of matrices U and V is commonly approximated numerically with methods such as gradient descent or alternating least squares. Recently, Lin [15] proposed an algorithm variation that resolves one of the convergence issues. His algorithm guarantees the convergence to a stationary point. However, Lin's algorithm requires even more execution time per iteration than the Lee and Seung [13] NMF algorithm. The basic gradient descent algorithm [3] for computing matrices U and V, which is based on the additive update rules, is shown in Fig. 2.1:

$U = \text{rand}(m,k)$; %initialize U
$V = \text{rand}(k,n)$; %initialize V
for $i = 1 : \text{maxiter}$
$U = U - \eta_U \frac{\partial}{\partial U}(\frac{1}{2}||A - UV||_F^2)$
$V = V - \eta_V \frac{\partial}{\partial V}(\frac{1}{2}||A - UV||_F^2)$
end

Fig. 2.1 Basic gradient descent algorithm for NMF

As shown, in Fig. 2.1, the gradient descent algorithm always takes a step in the direction of the negative gradient, i.e., the steepest descending direction of the function. Step size parameters η_U and η_V usually take a small value close to zero. Please notice that in order to prevent values of matrices U and V from becoming negative, after the application of each update rule, we set any derived negative value of matrices U and V to be equal to 0.

Finally, we will apply the NMF algorithm to our running example of Fig. 1.2 in Sect. 1.1 of Chap. 1. The resulting matrices after the application of NMF to the training data of our running example are shown in Fig. 2.2. Please notice that in Chap. 3, we will describe in detail the UV decomposition method, which is an instance of NMF without the constraint of nonnegativity of U and V matrices.

$$\underset{\hat{A}_{m\times n}}{\begin{bmatrix} 2.82 & 0.87 & 2.46 & 4.31 \\ 1.05 & 4.46 & 1.66 & 0.07 \\ 3.11 & 1.01 & 2.73 & 4.74 \\ 0.87 & 3.60 & 1.36 & 0.10 \end{bmatrix}} = \underset{U_{m\times k}}{\begin{bmatrix} 5.12 & 0.00 \\ 0.08 & 2.57 \\ 5.64 & 0.03 \\ 0.12 & 2.07 \end{bmatrix}} \times \underset{V^{\top}_{k\times n}}{\begin{bmatrix} 0.55 & 0.17 & 0.48 & 0.84 \\ 0.39 & 1.73 & 0.63 & 0.00 \end{bmatrix}}$$

Fig. 2.2 $\hat{A}_{m\times n}$ (initial matrix A) is the predicted matrix after the application of NMF algorithm

2.4 Latent Semantic Indexing

Furnas, Deerwester et al. [8] proposed the latent semantic indexing (LSI) in information retrieval to deal with high dimensionality of document-term matrix. More specifically, LSI uses SVD to capture latent associations between terms and documents. SVD is a well-known factorization technique that factors a matrix into three matrices. Document-term model and SVD are used in [8] for retrieving information. Documents and queries are represented with vectors and SVD is used for reducing these vector dimensions. Berry et al. [2] carried out a survey of computational requirements for managing (e.g., folding-in[1]) LSI-encoded databases.

[1]Folding in terms or documents is a simple technique that uses existing SVD to represent new information.

He claimed that the reduced-dimensions model is less noisy than the original data. Hofmann [12] proposed a model-based algorithm which relies on latent semantic and statistical models.

One main category of collaborative filtering (CF) algorithms in recommender systems are model-based algorithms, which recommend to first develop a model of user ratings for items. 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 nearest-neighbor CF algorithms in terms of accuracy with efficiency in terms of execution time. Toward this direction, LSI is a technique that has been extensively used in information retrieval. LSI detects latent relationships between documents and terms. In CF, LSI can be used to form users' trends from individual preferences, by detecting latent relationships between users and items. Therefore, with LSI, a higher level representation of the original user–item matrix is produced, which presents a threefold advantage: (i) It contains the main trends of users' preferences, (ii) noise is removed, and (iii) it is much more condensed than the original matrix thus it favors scalability.

A well-known latent factor model for matrix decomposition is SVD. The SVD [26] of a matrix $A_{I_1 \times I_2}$ can be written as a product of three matrices, as shown in Eq. 2.17:

$$A_{I_1 \times I_2} = U_{I_1 \times I_1} \cdot S_{I_1 \times I_2} \cdot {V^\top}_{I_2 \times I_2}, \tag{2.17}$$

where U is the matrix with left singular vectors of A, $V^\top$ is the transpose of the matrix V with right singular vectors of A, and S is the diagonal matrix of ordered singular values of A. Please note that singular values determined by the factorization of Eq. 2.17 are unique and satisfy $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \cdots \geq \sigma_{I_2} \geq 0$.

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 [8], to deal with latent semantic associations of terms in texts and to reveal major trends in A. Please notice that in Chap. 3, we will describe in detail the SVD decomposition method, which is the core method that LSI relies on.

2.5 Probabilistic Latent Semantic Indexing

Probabilistic latent semantic indexing (PLSI), which is the probabilistic case of the LSI technique, can also be used either for discovering main trends of a user–item rating matrix in recommender system domain or for discovering abstract "topics" that occur in a collection of documents in the information retrieval domain.

In PLSA for the information retrieval domain, it is assumed that there are k latent topics, stated as $T_1, \ldots, T_k$. PLSI predicts frequency values of a document-word matrix A in two steps. First the algorithm chooses a latent topic T_l with probability $P(T_l)$ and then it generates the indices (i, j) of the document-word matrix with probabilities $P(d_i|T_l)$ and $P(w_j|T_l)$, respectively. All terms used in the Probabilistic Latent Semantic Analysis (PLSA) algorithm, such as $P(T_l)$, $P(d_i|T_l)$, and $P(w_j|T_l)$, must be computed from observed frequencies in the document-term matrix A.

The aforementioned three key sets of parameters create an SVD-like matrix factorization of the $m \times n$ document-term matrix A. Therefore, the (i, j)th value of A can be interpreted as an "observed instantiation" of the probability $P(d_i, w_j)$. If U_k is the $m \times k$ matrix, in which the (i, l)th entry is $P(d_i|T_l)$, S_k is the $k \times k$ diagonal matrix in which the lth diagonal value is $P(T_l)$, and V_k is the $n \times k$ matrix in which (j, l)th entry is $P(w_j|T_l)$, then $P(d_i, w_j)$ of matrix A is computed as illustrated in Eq. 2.19.

$$R = U_k \times S_k \times V_k^T \tag{2.18}$$

$$P(d_i, w_j) = \sum_{l=1}^{k} P(d_i|T_l)P(T_l)P(w_j|T_l) \tag{2.19}$$

The left-hand side of Eq. 2.19 is the (i, j)th value of A, whereas the right-hand side is the (i, j)th entry of the product $U_k \times S_k \times V_k^T$. Depending on the number of topics k, the PLSA calculation can only be an approximation of the matrix A. PLSI supplies us with a different and quite strong technique for applying dimensionality reduction and has many advantages over its ancestor (LSI). One of its main advantages is that it performs a nonnegative matrix factorization, which makes it adequate for recommender systems, since it does not predict negative ratings. However, it also has some limitations. For example, the number of parameters grows linearly with the number of documents/items, which increases drastically the time/space complexity of the algorithm.

2.6 CUR Matrix Decomposition

In this section, we are going to present another matrix decomposition method, which is known as *CUR matrix decomposition*, because the initial matrix A is factorized to three matrices (C, U, and R). In high-dimensional data sets, several matrix decomposition methods, such as the SVD method, produce decomposed matrices which tend to be very dense, a fact that makes their processing a challenge in terms of efficiency. In contrast, the CUR decomposition method confronts this problem as it decomposes an original matrix into two sparse matrices C and R and only one dense matrix U, whose size is quite small. Moreover, CUR gives an exact decomposition no matter how many dimensions picked from the origin matrix (i.e., how big is parameter c),

whereas in SVD, the parameter c should be at least equal to the rank of the origin matrix A [7, 16, 20].

Next, we will describe briefly the process of CUR decomposition. Let A be a matrix with m rows and n columns and c the number of main dimensions that we want to keep after decomposing A. The basic components of CUR decomposition of A are the following three matrices:

1. Based on selected c columns of A, we produce the $m \times c$ matrix C.
2. Based also on selected c columns of A, we construct the $c \times n$ matrix R.
3. We build a $c \times c$ matrix U, which is constructed as follows:

   - First of all, we define a matrix W that is the intersection of chosen columns and rows of matrices C and R, respectively.
   - Next, we compute the SVD decomposition of W: $W = \tilde{U}\tilde{S}\tilde{V}^\top$ and we also compute the Moore–Penrose pseudoinverse $\tilde{S}^+$ of the diagonal matrix $\tilde{S}$. To do this, we replace every nonzero element s of the diagonal matrix $\tilde{S}$ with $\frac{1}{s}$.
   - Finally, matrix U is computed as follows: $U = \tilde{V}(\tilde{S}^+)^2\tilde{U}^\top$.

One quick observation about CUR decomposition is that row and column that are used to construct matrices C and R are randomly selected from matrix A. It is obvious that this selection will affect CUR approximation. For this reason, both the row and column selection process should be biased, in a way so that more important rows or columns have a better chance of being picked. To select the most important row/column of matrix A, we can use the square of the Frobenius norm of A, which is the square root of the sum of the absolute squares of the elements of a matrix.[2]

Then, we can define as $Prob(i)$ the probability for a row/column to be selected as follows:

$$Prob(i) = \sum_{j=1}^{n} \frac{a_{i,j}^2}{\|A\|_F} \tag{2.20}$$

We will apply Eq. 2.20 in the training data of our running example of Fig. 1.2 in Sect. 1.1 of Chap. 1, as shown in Table 2.1:

Table 2.1 The training data of our running example of matrix A with the calculated Frobenius norm per row/column

4	1	1	4	**34**
1	4	2	0	**21**
2	1	4	5	**46**
21	**18**	**21**	**41**	**202**

$A_{3\times4}$

The last column of Table 2.1 presents the sum of its row elements' squares. Respectively, the last row of Table 2.1 represents the sum of squares of elements

[2] Let A be an $m \times n$ matrix. So the Frobenius norm is $\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{i,j}|^2}$.

for each column of matrix A. Finally, the element (202) of the intersection of the fourth row and fifth column of Table 2.1 is the Frobenius norm of matrix A. Now we can define the probability $P(i)$ for every row or column. For example, the probability for the first row to be selected is $\frac{34}{202} = 0.16$. This means that there is a 16 % probability to select the first row of the matrix A.

Till now, we have shown how a column or row of A is being selected in CUR decomposition. Before it becomes a column/row of matrices C and R, respectively, we have to rescale it up accordingly. To do this, we divide each column's element of matrix A by the square root of the expected number of times this column would be picked (i.e., $\sqrt{c \times P(i)}$). Thus, if we set $c = 2$ and we then choose the first row with probability 0.16, the vector will be divided by $\sqrt{2 \times 0.16}$. The new scaled row would be [7.07 1.76 1.76 7.07]. The same procedure is followed for scaling all chosen columns of A, which become columns for matrices C and R, respectively.

Next, we will present the U's construction through an example. Let us assume that we chose the first and the second rows and the last two columns of matrix A of Table 2.1. In this way, matrix W would be equal to:

$$W = \begin{bmatrix} 1 & 4 \\ 2 & 0 \end{bmatrix}$$

Here is the SVD of W:

$$\underset{W}{\begin{bmatrix} 1 & 4 \\ 2 & 0 \end{bmatrix}} = \underset{\tilde{U}}{\begin{bmatrix} 0.99 & 0.15 \\ 0.15 & -0.99 \end{bmatrix}} \times \underset{\tilde{S}}{\begin{bmatrix} 4.16 & 0 \\ 0 & 1.92 \end{bmatrix}} \times \underset{\tilde{V}^\top}{\begin{bmatrix} 0.31 & 0.95 \\ -0.95 & -0.31 \end{bmatrix}}$$

That is, the three matrices on the right are $\tilde{U}$, $\tilde{S}$, and $\tilde{V}^\top$, respectively. Matrix $\tilde{S}$ has no zeros along the diagonal, so each element is replaced by its numerical inverse to get its Moore–Penrose pseudoinverse S^+ matrix, by replacing every nonzero element s of the diagonal matrix $\tilde{S}$ with $\frac{1}{s}$.

$$S^+ = \begin{bmatrix} 0.24 & 0 \\ 0 & 0.52 \end{bmatrix}$$

So finally,

$$U = \tilde{V} S^+ \tilde{U}^\top$$

$$\Longleftrightarrow$$

$$U = \begin{bmatrix} 0.31 & -0.95 \\ 0.95 & -0.31 \end{bmatrix} \times \begin{bmatrix} 0.06 & 0 \\ 0 & 0.27 \end{bmatrix} \times \begin{bmatrix} 0.99 & 0.15 \\ 0.15 & -0.99 \end{bmatrix} = \begin{bmatrix} -0.02 & 0.25 \\ 0.04 & 0.09 \end{bmatrix}$$

As we discussed earlier, approximation is only guaranteed to be close only when parameter c is large enough. However, the Frobenius norm helps us to select the

$$\underbrace{\begin{bmatrix} 3.94 & 10.35 & 5.35 & 1.44 \\ 1.85 & 9.70 & 4.77 & -0.61 \\ 7.47 & 26.29 & 13.26 & 0.96 \end{bmatrix}}_{\hat{A}} = \underbrace{\begin{bmatrix} 2.22 & 6.34 \\ 4.44 & 0.00 \\ 8.88 & 7.93 \end{bmatrix}}_{C} \times \underbrace{\begin{bmatrix} -0.02 & 0.25 \\ 0.04 & 0.52 \end{bmatrix}}_{U} \times \underbrace{\begin{bmatrix} 6.89 & 1.72 & 1.72 & 6.89 \\ 2.22 & 8.88 & 4.44 & 0 \end{bmatrix}}_{R}$$

Fig. 2.3 $\hat{A}_{4\times 3}$ (initial matrix A) is the predicted matrix after the application of CUR algorithm

most important rows and columns and as a result, CUR decomposition tends to have high effectiveness. For our running example, the resulting matrices after CUR decomposition are shown in Fig. 2.3.

In conclusion, when we apply the CUR method on a sparse matrix A, there is a main advantage over the SVD method. The two large matrices C and R, which are analogous to U and V of SVD method, control the sparsity property from the original matrix A. Only the matrix U (analogous to S) is dense, but this matrix is small, so the density does not affect the performance of an algorithm that would process the matrix data. However, we should emphasize the fact that CUR approximation matrices are less accurate than SVD approximation matrices. Lastly, since the rows and columns in CUR come from the original matrix (rather than left and right singular vectors as in SVD), the CUR approximation matrix is often easier for users to comprehend.

2.7 Other Matrix Decomposition Methods

In addition to methods presented in this chapter, there are many other interesting matrix decomposition approaches.

Sarwar et al. [22–24] applied dimensionality reduction for the user-based CF approach. He also used SVD for generating rating predictions. The case study of [17] presents two different experiments that compare the quality of a recommender system using SVD with the quality of a recommender system using naive collaborative filtering. The results suggest that SVD has the potential to meet many of the challenges of recommender systems, under certain conditions. However, in contrast to Symeonidis et al.'s [27–30] work, Sarwar et al. [23, 24] did not consider two significant issues: (i) Predictions should be based on the users' neighbors and not on the test (target) user, as the ratings of the latter are not a priori known. For this reason, Symeonidis et al. rely only on the neighborhood of the test user. (ii) The test users should not be included in the calculation of the model, because they are not known during factorization phase. For this reason, Symeonidis et al. introduced the notion of pseudo-user in order to include a new user in the model (folding-in), from which recommendations are derived. We have to mention that for high-dimensional data sets (i.e., with number of dimensions more than 1000), dimensionality reduction is usually performed prior to applying a *k-nearest neighbor* algorithm (k-NN). Please notice that for very high-dimensional data sets (i.e., live video streams, DNA data, or high-dimensional time series data), running a fast approximate k-NN search using

"locality-sensitive hashing," "random projections," and "sketches" might be the only feasible option [4, 25].

Other related work also includes Goldberg et al. [10], who applied PCA to facilitate offline dimensionality reduction for clustering the users, and therefore manages to have rapid online computation of recommendations. Hofmann [12] proposed a model-based algorithm which relies on latent semantic and statistical models. Moreover, the authors in [9] propose a method that creates self-similarity matrices from eigenvectors calculated by SVD, in a way that separates the concepts. Then, these matrices are combined into a new matrix by applying an aggregating function.

Finally, another decomposition approach, known as Cholesky decomposition, decomposes a $n \times n$ symmetric, positive-definite matrix A into the product of a lower triangular matrix with positive diagonal elements B and its conjugate transpose B^* [32]

$$A = BB^*, \tag{2.21}$$

Please notice that if A is not symmetric and positive definite, then the algorithm may have a zero entry in the diagonal of B. Cholesky factorization is an efficient algorithm, which is faster than other similar decomposition algorithms and produces good approximations of the initial matrix A.

References

1. Bensmail, H., Celeux, G.: Regularized gaussian discriminant analysis through eigenvalue decomposition. J. Am. Stat. Assoc. **91**(436), 1743–1748 (1996)
2. Berry, M., Dumais, S., O'Brien, G.: Using linear algebra for intelligent information retrieval. SIAM Rev. **37**(4), 573–595 (1994)
3. Berry, M.W., Browne, M., Langville, A.N., Paul Pauca, V., Plemmons, R.J.: Algorithms and applications for approximate nonnegative matrix factorization. Comput. Stat. Data Anal. **52**(1), 155–173 (2007)
4. Beyer, K.S., Goldstein, J., Ramakrishnan, R., Shaft, U.: When is "nearest neighbor" meaningful? In: ICDT, Lecture Notes in Computer Science, vol. 1540, pp. 217–235. Springer (1999)
5. Dhillon, I.S., Sra, S.: Generalized nonnegative matrix approximations with bregman divergences. In: NIPS, pp. 283–290 (2005)
6. Dhillon, I.S., Sra, S.: Generalized nonnegative matrix approximations with bregman divergences. In: Neural Information Processing Systems, pp. 283–290 (2005)
7. Drineas, P., Kannan, R., Mahoney, M.W.: Fast monte carlo algorithms for matrices III: computing a compressed approximate matrix decomposition. SIAM J. Comput. **36**(1), 184–206 (2006)
8. 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)
9. Gabriel, H.-H., Spiliopoulou, M., Nanopoulos, A.: Eigenvector-based clustering using aggregated similarity matrices. In: Proceedings of the 2010 ACM Symposium on Applied Computing, pp. 1083–1087. ACM (2010)
10. Goldberg, K., Roeder, T., Gupta, T., Perkins, C.: Eigentaste: a constant time collaborative filtering algorithm. Inf. Retrieval **4**(2), 133–151 (2001)
11. Golub, G.H., Van Loan, C.F.: Matrix Computations, 3rd edn. Johns Hopkins University Press, Baltimore, MD, USA (1996)

12. Hofmann, T.: Latent semantic models for collaborative filtering. ACM Trans. Inf. Syst. **22**(1), 89–115 (2004)
13. Lee, D.D., Seung, H.S.: Learning the parts of objects by nonnegative matrix factorization. Nature **401**, 788–791 (1999)
14. Lee, D.D., Seung, H.S.: Algorithms for non-negative matrix factorization. In: NIPS, pp. 556–562 (2000)
15. Lin, C.-J.: On the convergence of multiplicative update algorithms for nonnegative matrix factorization. IEEE Trans. Neural Netw. **18**(6), 1589–1596 (2007)
16. Mahoney, M.W., Maggioni, M., Drineas, P.: Tensor-cur decompositions for tensor-based data. SIAM J. Matrix Anal. Appl. **30**(3), 957–987 (2008)
17. Mirza, B.J.: Jumping connections: a graph-theoretic model for recommender systems. Technical report, Master's thesis, Virginia Tech (2001)
18. Moler, C.B., Stewart, G.W.: An algorithm for generalized matrix eigenvalue problems. SIAM J. Numer. Anal. **10**(2), 241–256 (1973)
19. Pudil, P., Novovičová, J.: Novel methods for feature subset selection with respect to problem knowledge. In: Feature Extraction. Construction and Selection. Springer International Series in Engineering and Computer Science, vol. 453, pp. 101–116. Springer, US (1998)
20. Anand Rajaraman and Jeffrey David Ullman: Mining of Massive Datasets. Cambridge University Press, New York, NY, USA (2011)
21. Roweis, S.T., Saul, L.K.: Nonlinear dimensionality reduction by locally linear embedding. Science **290**(5500), 2323–2326 (2000)
22. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Analysis of recommendation algorithms for e-commerce. In: Proceedings of ACM Electronic Commerce Conference, pp. 158–167 (2000)
23. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Application of dimensionality reduction in recommender system-a case study. In: ACM WebKDD Workshop (2000)
24. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Incremental singular value decomposition algorithms for higly scalable recommender systems. In: Proceedings of Computer and Information Technology (2002)
25. Shaw, B., Jebara, T.: Structure preserving embedding. In: Proceedings of the 26th Annual International Conference on Machine Learning, ICML'09, New York, NY, USA, pp. 937–944. ACM (2009)
26. Strang, G.: Linear Algebra and Its Applications. Thomson Brooks/Cole (2006)
27. Symeonidis, P.: Content-based dimensionality reduction for recommender systems. In: Proceedings of the 31st Conference of the German Classification Society (GfKl'2007), Freiburg (2007)
28. Symeonidis, P., Nanopoulos, A., Papadopoulos, A., Manolopoulos, Y.: Collaborative filtering based on users trends. In: Proceedings of the 30th Conference of the German Classification Society (GfKl'2006), Berlin (2006)
29. Symeonidis, P., Nanopoulos, A., Papadopoulos, A., Manolopoulos, Y.: Scalable collaborative filtering based on latent semantic indexing. In: Proceedings of the 21st AAAI Workshop on Intelligent Techniques for Web Personalization (ITWP), pp. 1–9, Boston, MA (2006)
30. Symeonidis, P., Nanopoulos, A., Papadopoulos, A., Manolopoulos, Y.: Collaborative recommender systems: combining effectiveness and efficiency. Expert Syst. Appl. **34**(4), 2995–3013 (2008)
31. Wikipedia: Non-negative matrix factorization—wikipedia, the free encyclopedia, 2015. Accessed 13 Sept 2015
32. Wilkinson, J.H., Wilkinson, J.H., Wilkinson, J.H.: The Algebraic Eigenvalue Problem, vol. 87. Clarendon Press Oxford (1965)

Chapter 3
Performing SVD on Matrices and Its Extensions

Abstract In this chapter, we describe singular value decomposition (SVD), which is applied on recommender systems. We discuss in detail the method's mathematical background and present (step by step) the SVD method using a toy example of a recommender system. We also describe in detail UV decomposition. This method is an instance of SVD, as we mathematically prove. We minimize an objective function, which captures the error between the predicted and real value of a user's rating. We also provide a step-by-step implementation of UV decomposition using a toy example, which is followed by a short representation of the algorithm in pseudocode form. Finally, an additional constraint of friendship is added to the objective function to leverage the quality of recommendations.

Keywords Singular value decomposition · UV decomposition

3.1 Singular Value Decomposition (SVD)

Singular value decomposition (SVD) is a very important linear algebra tool that we use to solve many mathematical problems. The SVD method is a factorization of a real or complex matrix [1]. In this section, we present the mathematical formulation of SVD and some of its variations.

First of all, we present SVD's connection with the eigenvalue decomposition method, which is described in Sect. 2.2. The relation between SVD and the eigenvalue decomposition method comes from a special case of the latter one which will be analyzed subsequently.

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_3

If and only if, a matrix A is symmetric and positive definite (i.e., $A \iff A = A^\top \quad \wedge \quad \forall \mathbf{e} \in E, \mathbf{e} > 0$), then SVD and eigenvalue decomposition both coincide as follows:

$$A = USU^\top = E\Lambda E^{-1} \tag{3.1}$$

with $U = E$ and $S = \Lambda$.

In case that matrix A is a nonsquare matrix and its factorization can be written as $A = USV^\top$, then there are two other matrices $M_1 = A^\top A$ and $M_2 = AA^\top$ and their factorization, which is of special interest:

$$\begin{gathered}
M_1 = A^\top A \iff \\
M_1 = (USV^\top)^\top (USV^\top) \iff \\
M_1 = (VS^\top U^\top)(USV^\top) \iff \xrightarrow{U\text{-}unitary} \\
M_1 = VS^\top ISV^\top \iff \\
M_1 = VS^\top SV^\top \iff \xrightarrow{S\text{-}orthogonal} \\
M_1 = A^\top A = VS^2 V^\top \\
\wedge \\
M_2 = AA^\top \iff \\
M_2 = (USV^\top)(USV^\top)^\top \iff \\
M_2 = (USV^\top)(VS^\top U^\top) \iff \xrightarrow{V\text{-}unitary} \\
M_2 = USIS^\top U^\top \iff \\
M_2 = USS^\top U^\top \iff \xrightarrow{S\text{-}orthogonal} \\
M_2 = AA^\top = US^2 U^\top
\end{gathered} \tag{3.2, 3.3}$$

Please notice that for both matrices (M_1 and M_2), the application of SVD on the original matrix A can be used to compute their own SVD factorization. And since these matrices are by definition symmetric and positive definite, this is also their eigenvalue decomposition, with eigenvalues $\Lambda = S^2$. For example, we can calculate SVD decomposition of M_1 matrix as follows:

$$M_1 = A^\top A = VS^2 V^\top \tag{3.4}$$

The S matrix holds in its diagonal the eigenvalues of A, and matrices A and $A^\top$ are known. So, we can compute matrix V from Eq. 3.4. It remains to compute only matrix U. Its computation can be done, using the basic equation of the SVD method as follows:

$$
\begin{aligned}
A &= USV^\top \iff \\
AV &= USVV^\top \iff \xrightarrow{V\text{-}unitary} \\
AV &= USI \iff \\
AV &= US \iff \\
AVS^{-1} &= USS^{-1} \iff \\
AVS^{-1} &= UI \iff \\
AVS^{-1} &= U
\end{aligned} \tag{3.5}
$$

Based on Eq. 3.5, we can compute matrix U and now we have known all parts to apply the SVD algorithm on matrix A.

Analogously, we can compute matrix V using M_2 matrix. The explanation is given subsequently:

$$
\begin{aligned}
M_2 &= AA^\top = US^2U^\top \\
&\wedge \\
A &= USV^\top \iff \\
U^\top A &= U^\top USV^\top \iff \xrightarrow{U\text{-}unitary} \\
U^\top A &= ISV^\top \iff \\
U^\top A &= SV^\top \iff \\
S^{-1}U^\top A &= SS^{-1}V^\top \iff \\
S^{-1}U^\top A &= IV^\top \iff \\
S^{-1}U^\top A &= V^\top
\end{aligned} \tag{3.6, 3.7}
$$

A reasonable question arises according to the above calculation methods. *When do we use matrix M_1 and when matrix M_2?* The answer is simple. We choose the minimum dimension of the matrix A. With mathematical explanation, if matrix A is an $n \times m$ matrix, we choose M_1 matrix when $m \ll n$ and M_2 matrix when $n \ll m$. Let us give an example to make it more clear. If matrix A is a 1.000×100 matrix, we prefer to use M_1 matrix which will be a square matrix with 100×100 dimensions. On the other hand, if matrix A is a 100×1.000 matrix, we choose to use the matrix M_2 with its dimensions 100×100 also. This selection saves data space and makes the *SVD* calculation easily and faster.

In this point, we have to analyze and represent more particularly the implementation of the SVD method. Formally, applying SVD on a matrix A means that A can be written as a product of three matrices, as shown in Eq. 3.9:

$$
A_{m \times n} = U_{m \times m} \cdot S_{m \times n} \cdot V^T_{n \times n} \tag{3.8}
$$

where U is an $m \times m$ real or complex unitary matrix, S is an $m \times n$ rectangular diagonal matrix with nonnegative real numbers on the diagonal, and $V^\top$ (the conjugate

transpose of matrix V, or simply the transpose of V if V is real) is an $n \times n$ real or complex unitary matrix. The diagonal entries $S_{i,i}$ of matrix S are known as singular values of A. The m columns of matrix U and the n columns of matrix V are called left singular vectors and right singular vectors of A, respectively.

The singular value decomposition and eigenvalue decomposition are closely related. Namely:

- The left singular vectors of A are eigenvectors of $A \cdot A^\top$.
- The right singular vectors of A are eigenvectors of $A^\top \cdot A$.
- The nonzero singular values of A (found on diagonal entries of S) are square roots of nonzero eigenvalues of both $A^\top \cdot A$ and $A \cdot A^\top$.

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 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 the main diagonal of matrix S (also known as nuclear norm). Next, we preserve 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 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 current ratings of the target user u, we enter pseudo-user vector in c-dimensional space. Finally, we find k-nearest neighbors of pseudo-user vector in c-dimensional space and apply either user- or item-based similarity to compute the top-N recommended items. Conclusively, provided recommendations consider the existence of user-rating trends, as similarities are computed in the reduced c-dimensional space, where dimensions correspond to trends.

To ease the discussion, we will apply the SVD algorithm on our running example of Fig. 1.2 in Sect. 1.1 of Chap. 1. The same figure is represented again below in Fig. 3.1 for convenience. As shown, the example data set is divided into training and test sets. The null cells (no rating) are presented as zeros.

Fig. 3.1 **a** Training set (3×4), **b** Test set (1×4)

(a)

	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

(b)

	I_1	I_2	I_3	I_4
U_4	1	4	1	0

3.1.1 Applying the SVD and Preserving the Largest Singular Values

Initially, we apply SVD to an $n \times m$ matrix A (i.e., the training data of our running example) that produces the decomposition shown in Eq. 3.9. The matrices of our running example are shown in Fig. 3.2.

$$A_{n\times m} = U_{n\times n} \cdot S_{n\times m} \cdot V^{\top}{}_{m\times m} \ . \tag{3.9}$$

$$\underset{A_{n\times m}}{\begin{bmatrix} 4 & 1 & 1 & 4 \\ 1 & 4 & 2 & 0 \\ 2 & 1 & 4 & 5 \end{bmatrix}} = \underset{U_{n\times n}}{\begin{bmatrix} -0.61 & 0.28 & -0.74 \\ -0.29 & -0.95 & -0.12 \\ -0.74 & 0.14 & 0.66 \end{bmatrix}} \times \underset{S_{n\times m}}{\begin{bmatrix} 8.87 & 0.00 & 0.00 & 0.00 \\ 0.00 & 4.01 & 0.00 & 0.00 \\ 0.00 & 0.00 & 2.51 & 0.00 \end{bmatrix}}$$

$$\times \underset{V^{\top}{}_{m\times m}}{\begin{bmatrix} -0.47 & -0.28 & -0.47 & -0.69 \\ 0.11 & -0.85 & -0.27 & 0.45 \\ -0.71 & -0.23 & 0.66 & 0.13 \\ -0.52 & 0.39 & -0.53 & 0.55 \end{bmatrix}}$$

Fig. 3.2 Example of $A_{n\times m}$ (initial matrix A), $U_{n\times m}$ (left singular vectors of A), $S_{n\times m}$ (singular values of A), $V\top_{m\times m}$ (right singular vectors of A)

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 Eq. (3.10) and Fig. 3.3:

$$\hat{A}_{n\times m} = U_{n\times c} \cdot S_{c\times c} \cdot V^{\top}{}_{c\times m} \ . \tag{3.10}$$

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 to a distinctive rating trend. We have to notice that in the running example, we create a two-dimensional space using

$$\begin{bmatrix} 2.69 & 0.57 & 2.22 & 4.25 \\ 0.78 & 3.93 & 2.21 & 0.04 \\ 3.17 & 1.38 & 2.92 & 4.78 \end{bmatrix} = \begin{bmatrix} -0.61 & 0.28 \\ -0.29 & -0.95 \\ -0.74 & 0.14 \end{bmatrix} \times \begin{bmatrix} 8.87 & 0.00 \\ 0.00 & 4.01 \end{bmatrix}$$

$$\hat{A}_{n\times i} \qquad U_{n\times c} \qquad S_{c\times c}$$

$$\times \begin{bmatrix} -0.47 & -0.28 & -0.47 & -0.69 \\ 0.11 & -0.85 & -0.27 & 0.45 \end{bmatrix}$$

$$V^{\top}{}_{c\times m}$$

Fig. 3.3 Example of $\hat{A}_{n\times m}$ (approximation matrix of A), $U_{n\times c}$ (left singular vectors of $\hat{A}$), $S_{c\times c}$ (singular values of $\hat{A}$), $V'_{c\times m}$ (right singular vectors of $\hat{A}$)

83.7 % of the total information of the matrix (12.88/15.39). Please note that the number 15.39 is the sum of elements in the main diagonal of $S_{c\times c}$ (singular values of $\hat{A}$).

3.1.2 Generating the Neighborhood of Users/Items

Having a reduced dimensional representation of the original space, we form the neighborhoods of users/items in that space. Please note that the original space consists of two subspaces:

- range of (A) whose U (see Fig. 3.3) is an orthonormal basis. This vector space is the column space of A referred to users.
- range of ($A^{\top}$) whose V (see Fig. 3.3) is an orthonormal basis. This vector space is the row space of A referred to items.

In particular, there are 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 $A^{\top}$, 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 rating 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, 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.

3.1.3 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) items according to their frequency in the found neighborhood of the target user and recommend the first N of them.

As another method, someone could use predicted values for each item to rank them. This ranking criterion, denoted as highest predicted rated item recommendation (HPR), is influenced by the mean absolute error (MAE[1]) between the predicted and real preferences of a user for an item. HPR opts for recommending the items that are more probable to receive a higher rating. Notice that HPR shows poor performance for classic collaborative filtering (CF) algorithms. However, it shows very good results when it is used in combination with SVD. The reason is that in the latter, it is based only on major trends of users.

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

3.1.4 Inserting a Test User in the c-Dimensional Space

Related work [5] has studied SVD on CF considering the test data as a priori known. It is evident that, for the 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 current ratings of the test user u, we enter a pseudo-user vector in the c-dimensional space using Eq. (3.11) [4]. In the current example, we insert U_4 into two-dimensional space, as it is shown in Fig. 3.4:

$$\mathbf{u}_{new} = \mathbf{u} \cdot V_{m \times c} \cdot {S^{-1}}_{c \times c} \tag{3.11}$$

[1] $MAE = \frac{1}{n} \sum_{i=1}^{n} |f_i - y_i|$: 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.

$$\begin{bmatrix}-0.23 & -0.89\end{bmatrix} = \begin{bmatrix}1 & 4 & 1 & 0\end{bmatrix} \times \begin{bmatrix}-0.47 & 0.11\\ -0.28 & -0.85\\ -0.47 & -0.27\\ -0.69 & 0.45\end{bmatrix} \times \begin{bmatrix}0.11 & 0.00\\ 0.00 & 0.25\end{bmatrix}$$

$\mathbf{u}_{new}$ $\mathbf{u}$ $V_{m\times c}$ ${S^{-1}}_{c\times c}$

Fig. 3.4 Example of $\mathbf{u}_{new}$ (inserted new user vector), $\mathbf{u}$ (user vector), $\mathbf{V}_{m\times c}$ (two *right* singular vectors of V), ${S^{-1}}_{c\times c}$ (two singular values of inverse S)

In Eq. (3.11), $\mathbf{u}_{new}$ denotes 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 Fig. 3.3.

Notice that the inserted vector values of test user U_4 are very similar to those of U_2 after the insertion, as shown in Fig. 3.5.

Fig. 3.5 The new $U_{n+1,c}$ matrix containing the new user ($\mathbf{u}_{new}$) that we have added

$$\begin{bmatrix}-0.61 & 0.28\\ \mathbf{-0.29} & \mathbf{-0.95}\\ -0.74 & 0.14\\ \mathbf{-0.23} & \mathbf{-0.89}\end{bmatrix}$$

This is reasonable, because these two users have similar ratings as it is shown in Fig. 3.1a, b.

3.2 From SVD to UV Decomposition

In this section, we present the theoretical and mathematical background of a *UV decomposition* algorithm. We will use the *UV decomposition* method to predict the blank/zero entries in the user–item rating matrix A. First, we will make predictions based only on the *user–item* rating matrix. Then, we will try to improve our rating predictions by exploiting also additional resources (i.e., the friendship network of users). The *UV decomposition* can be considered as an instance of *SVD*, which is described as follows:

$$A = \tilde{U} \cdot \tilde{S} \cdot \tilde{V^{\top}} \tag{3.12}$$

This claim can be mathematically proved using the properties of linear regression. Let us assume that:

- $\mathcal{U} = \tilde{U}$ where $\tilde{U}$ holds the top-k left singular vectors of A.
- $\mathcal{V} = \tilde{S}\tilde{V}^{\top}$ where $\tilde{S} = diag(\sigma_1, \ldots, \sigma_k)$ and $\tilde{V}$ holds the top-k right singular vectors of A.

More particularly, linear regression searches for two matrices $\mathcal{U} \in \mathbb{R}^{m \times k}$ and $\mathcal{V} \in \mathbb{R}^{k \times n}$, where $k \leq rank(A)$ to minimize the following equation:

$$minimize_{\mathcal{U}^\top \mathcal{U} = I_k} \|A - \mathcal{U}\mathcal{V}\|_F^2 \tag{3.13}$$

Note that matrix $\mathcal{U}$ should be orthogonal for convention.

Given a vector $\mathbf{a} \in \mathbb{R}^m$ and a matrix $\mathcal{U} \in \mathbb{R}^{m \times d}$, linear regression finds a coefficient vector $\mathbf{v} \in \mathbb{R}^d$ that approximates a linearly based on $\mathcal{U}$. The vector $\mathbf{v}$ minimizes $\|\mathbf{a} - \mathcal{U}\mathbf{v}\|_2^2$ by assuming that $rank(\mathcal{U}) = d$, such that $\mathbf{v}$ can be expressed as:

$$\mathbf{v} = (\mathcal{U}^\top \mathcal{U})^{-1} \mathcal{U}^\top \mathbf{a} \tag{3.14}$$

Based on the Frobenius norm, we can rewrite Eq. 3.14 as:

$$\mathbf{v}_i = (\mathcal{U}^\top \mathcal{U})^{-1} \mathcal{U}^\top \mathbf{a}_i \tag{3.15}$$

and hence

$$\begin{gathered} \mathcal{V} = (\tilde{U}^\top \tilde{U})^{-1} \tilde{U}^\top A \iff \\ \mathcal{V} = (\tilde{U}^\top \tilde{U})^{-1} \tilde{U}^\top U S V^\top \iff \\ \mathcal{V} = I_k^{-1} (\tilde{U}^\top U) S V^\top \iff \\ \mathcal{V} = I_k^{-1} [I_k \quad 0] S V^\top \iff \\ \mathcal{V} = S V^\top \end{gathered} \tag{3.16}$$

Based on Eq. 3.16, we can rewrite *SVD* and Eq. 3.12 as follows:

$$A = \mathcal{U}\mathcal{V} \tag{3.17}$$

Based on the above proof, there is a matrix $\mathcal{V}$, which is the product of S and $V^\top$. In other words, S is left-blended into matrix $V^\top$ and produces matrix $\mathcal{V}$. We can assume that the new matrix $\mathcal{V}$ contains both the information on eigenvalues of the utility matrix A and the information of the matrix V. For the UV decomposition method, henceforth, we will use the same symbols U and V for factorized matrices of the method and there will be no relationship with U, V matrices of the SVD method. An easy way to estimate blank/zero entries in the user–item rating matrix A is to conjecture that the utility matrix is actually the product of two long, thin matrices U and V. To ease the discussion, we will present (step by step) the UV decomposition method on the user–item rating matrix A of our running example of Fig. 1.2 in Sect. 1.1 of Chap. 1. Please notice that in this case, we merge the training set of Fig. 1.2a with the test set of Fig. 1.2b. This merging results in a matrix A with 4 rows and 4 columns. The user–item rating matrix A of our running example has now $|I|$ rows and $|J|$ columns (size $|I| \times |J| = 4 \times 4$) and want to find a matrix U (with the size of $|I| \times K = 4 \times K$) and a matrix V (with the size of $|J| \times K = 4 \times K$),

so that their product approximates A:

$$A \approx UV^\top = \hat{A} \tag{3.18}$$

More generally, if we have a matrix A with n rows and m columns, we can find two matrices: matrix U with n rows and k columns and matrix V with m rows and k columns, such that $UV^\top$ approximates A, where most of its blank/zero entries are filled and there is a small divergence among its initial and predicted values of A (Fig. 3.6).

$$\underset{A_{m\times n}}{\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}} = \underset{U_{m\times k}}{\begin{bmatrix} u_{11} & \cdots & u_{1k} \\ \vdots & \ddots & \vdots \\ u_{m1} & \cdots & u_{mk} \end{bmatrix}} \times \underset{V^\top_{k\times n}}{\begin{bmatrix} v_{11} & \cdots & v_{1n} \\ \vdots & \ddots & \vdots \\ v_{k1} & \cdots & v_{kn} \end{bmatrix}}$$

Fig. 3.6 The general form of the *UV decomposition*. The initial matrix A can be presented as a product of two (smaller than A in dimensions) matrices U and V

To get the prediction of a rating that a user would give on an item, we can calculate the dot product of the two vectors, which correspond to $\mathbf{u}_i$ and $\mathbf{v}_j$:

$$\hat{a}_{ij} = \mathbf{u}_i \mathbf{v}_j^\top = \sum_{k=1}^{K} u_{ik} v_{kj} \tag{3.19}$$

The next step of the method is to find a way to obtain U and V. One way to solve the problem is to initialize the two matrices with some random values and to calculate how "different" is their product compared with A. Then, we can try to minimize this difference, iteratively. Such a numerical approximation method is called *gradient descent*, aiming to find a local minimum of the difference. The difference actually is the error between the real rating and the predicted one and can be computed for each user–item pair using Eq. 3.20:

$$e_{ij}^2 = (a_{ij} - \hat{a}_{ij})^2 = (a_{ij} - \sum_{k=1}^{K} u_{ik} v_{kj})^2 \tag{3.20}$$

Please notice that we compute the squared error in Eq. 3.20, because the predicted rating can be either higher or lower than the real rating. In addition, by computing the square of the error, we "reward" the small errors and "penalize" the greater ones. Figure 3.7 visualizes function $f(x) = x^2$.

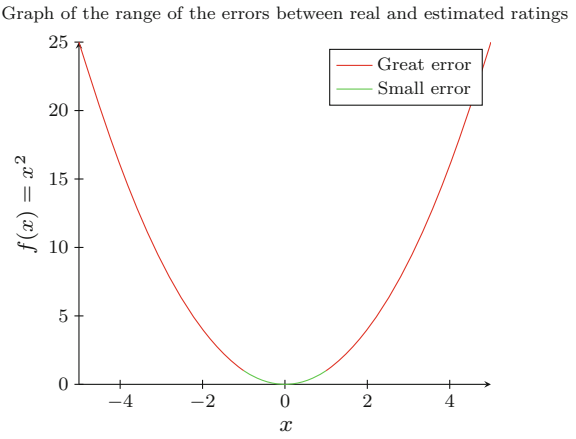

Fig. 3.7 Using function $f(x) = x^2$, we "reward" small errors and "penalize" great ones

As shown in Fig. 3.7, the green line part is the area that the square error is small and we "reward" these differences against the red-colored area where the error is significant and we "penalize" these differences.

3.2.1 Objective Function Formulation

In this section, we formulate the objective function, to present the additive update rules. As we have already described in Sect. 3.2, we formulate our objective function as follows:

$$G = \|A - \hat{A}\|_F^2 = \|A - UV\|_F^2 = e_{ij}^2 = (a_{ij} - \sum_{k=1}^{K} u_{ik} v_{kj})^2 \tag{3.21}$$

As shown by Eq. 3.21, our objective function equals the square error function of Eq. 3.20. So the challenge is to minimize the error among the real and predicted ratings of the user–item rating matrix A. Since there is no closed form solution for function G, we use a numerical approximation method, such as *gradient descent*, to solve it. Gradient descent finds the direction of change for each rating of matrix A, which most decreases the error function. In particular, to minimize the error function, we have to know in which direction to modify the values of u_{ik} and v_{kj} of Eq. 3.21. In other words, to minimize the error function, the partial derivatives of G with respect to U and V are computed:

$$\frac{\partial G}{\partial U} = \frac{\partial}{\partial u_{ik}} e_{ij}^2 = 2(a_{ij} - \hat{a}_{ij}) \frac{\partial}{\partial u_{ik}} (a_{ij} - \hat{a}_{ij}) \iff$$
$$\frac{\partial G}{\partial U} = \frac{\partial}{\partial u_{ik}} e_{ij}^2 = 2(a_{ij} - \hat{a}_{ij})(-v_{kj}) \iff$$
$$\frac{\partial G}{\partial U} = \frac{\partial}{\partial u_{ik}} e_{ij}^2 = -2e_{ij} v_{kj} \tag{3.22}$$

$$\wedge$$

$$\frac{\partial G}{\partial V} = \frac{\partial}{\partial v_{kj}} e_{ij}^2 = 2(a_{ij} - \hat{a}_{ij}) \frac{\partial}{\partial v_{kj}} (a_{ij} - \hat{a}_{ij}) \iff$$
$$\frac{\partial G}{\partial V} = \frac{\partial}{\partial v_{kj}} e_{ij}^2 = 2(a_{ij} - \hat{a}_{ij})(-u_{ik}) \iff$$
$$\frac{\partial G}{\partial V} = \frac{\partial}{\partial v_{kj}} e_{ij}^2 = -2e_{ij} u_{ik} \tag{3.23}$$

Having obtained the gradient, we can now formulate additive update rules for both u_{ik} and v_{kj}:

$$u'_{ik} = u_{ik} - \eta \frac{\partial}{\partial u_{ik}} e_{ij}^2 = u_{ik} + 2\eta e_{ij} v_{kj} \tag{3.24}$$

$$\wedge$$

$$v'_{kj} = v_{kj} - \eta \frac{\partial}{\partial v_{kj}} e_{ij}^2 = v_{kj} + 2\eta e_{ij} u_{ik} \tag{3.25}$$

Here, η is a constant whose value determines the rate of approaching the minimum. Usually, we choose a small step value for η such as 0.0002. The reason is that if we make a large step toward the minimum, we may run the risk of missing the minimum and end up oscillating around the minimum.

3.2.2 Avoiding Overfitting with Regularization

A common problem in prediction modeling is overfitting. That is, a prediction model is typically trained and maximizes its performance based on some set of training data. However, its effectiveness is determined not by its performance on the training data but by its ability to perform well on unseen/test data. The *overfitting* problem occurs when a model begins to "memorize" training data rather than "learning" to generalize from the trend of training data. We can avoid overfitting, by adding to our objective function a parameter β and modifying the squared error as follows:

$$G = e_{ij}^2 = (a_{ij} - \sum_{k=1}^{K} u_{ik} v_{kj})^2 + \frac{\beta}{2} \sum_{k=1}^{K} (\|U\|^2 + \|V\|^2) \tag{3.26}$$

The new parameter β is used to control magnitudes of user-feature and item-feature vectors. In particular, it downsizes the possible large number values of one vector, so that they cannot dominate the possible smaller number values of the other vector. In practice, β is set to some values in the range of 0.02. To minimize the extended error function, again partial derivatives of G with respect to U and V should be recomputed:

$$\begin{aligned}
\frac{\partial G}{\partial U} &= \frac{\partial}{\partial u_{ik}} e_{ij}^2 = 2(a_{ij} - \hat{a}_{ij}) \frac{\partial}{\partial u_{ik}} (a_{ij} - \hat{a}_{ij}) + \beta u_{ik} \iff \\
\frac{\partial G}{\partial U} &= \frac{\partial}{\partial u_{ik}} e_{ij}^2 = 2(a_{ij} - \hat{a}_{ij})(-v_{kj}) + \beta u_{ik} \iff \\
\frac{\partial G}{\partial U} &= \frac{\partial}{\partial u_{ik}} e_{ij}^2 = -2e_{ij} v_{kj} + \beta u_{ik}
\end{aligned} \tag{3.27}$$

$$\wedge$$

$$\begin{aligned}
\frac{\partial G}{\partial V} &= \frac{\partial}{\partial v_{kj}} e_{ij}^2 = 2(a_{ij} - \hat{a}_{ij}) \frac{\partial}{\partial v_{kj}} (a_{ij} - \hat{a}_{ij}) + \beta v_{kj} \iff \\
\frac{\partial G}{\partial V} &= \frac{\partial}{\partial v_{kj}} e_{ij}^2 = 2(a_{ij} - \hat{a}_{ij})(-u_{ik}) + \beta v_{kj} \iff \\
\frac{\partial G}{\partial V} &= \frac{\partial}{\partial v_{kj}} e_{ij}^2 = -2e_{ij} u_{ik} + \beta v_{kj}
\end{aligned} \tag{3.28}$$

The new additive update rules for both u_{ik} and v_{kj} are as follows:

$$u'_{ik} = u_{ik} - \eta \frac{\partial}{\partial u_{ik}} e_{ij}^2 = u_{ik} + \eta(2e_{ij} v_{kj} - \beta u_{ik}) \tag{3.29}$$

$$\wedge$$

$$v'_{kj} = v_{kj} - \eta \frac{\partial}{\partial v_{kj}} e_{ij}^2 = v_{kj} + \eta(2e_{ij} u_{ik} - \beta v_{kj}) \tag{3.30}$$

3.2.3 Incremental Computation of UV Decomposition

In this section, we apply incrementally the UV decomposition method in our running example of Fig. 1.2 in Chap. 1. Please notice that in this case, we merge the training set of Fig. 1.2a with the test set of Fig. 1.2b. That is, the test user is included in the calculation of UV decomposition, for reasons of brevity. The new merged initial user-rating matrix A of our running example is shown in Fig. 3.8. As it is shown in

Fig. 3.8, we decompose A into a 4-by-2 and a 2-by-4 matrix, U and V, respectively. This means that we create a two-dimensional feature space ($k = 2$).

Fig. 3.8 Applying the *UV decomposition* to our example with $K = 2$ and trying to figure out U and V matrices

$$\underset{A_{4\times 4}}{\begin{bmatrix} 4 & 1 & 1 & 4 \\ 1 & 4 & 2 & 0 \\ 2 & 1 & 4 & 5 \\ 1 & 4 & 1 & ? \end{bmatrix}} = \underset{U_{4\times 2}}{\begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \\ u_{31} & u_{32} \\ u_{41} & u_{42} \end{bmatrix}} \times \underset{V^{\top}_{2\times 4}}{\begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} \\ v_{21} & v_{22} & v_{23} & v_{24} \end{bmatrix}}$$

UV decomposition starts with some arbitrarily chosen U and V, and then repeatedly adjusts U and V to decrease the root-mean-square error (RMSE), which is shown in Table 4.1.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=1}^{m}(a_{ij} - \hat{a}_{ij})^2}{D}}, \quad \forall i \in n \; and \; j \in m : a_{ij} \neq 0 \tag{3.31}$$

where D is the sum of the nonzero entries in matrix A. Let us now assume, that in our running example, we set equal to 1 all initial elements of U and V matrices, whereas their product creates matrix $\hat{A}$, which holds predicted ratings, as it is shown in Fig. 3.9.

Fig. 3.9 Matrices U and V with all elements equal to 1

$$\underset{\hat{A}_{4\times 4}}{\begin{bmatrix} 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \end{bmatrix}} = \underset{U_{4\times 2}}{\begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}} \times \underset{V^{\top}_{2\times 4}}{\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}}$$

Then, we can compute the RMSE for A and $\hat{A}$ matrices of Figs. 3.8 and 3.9, respectively, based on Eq. 3.31. For instance, to compute RMSE for the first rows of A and $\hat{A}$ matrices, we subtract 2 from each of the element in the first row of A, to get 2,−1, −1, and 2. We square and sum these to get 10. In the second row, the last column is zero, so this element is ignored when computing RMSE. The differences are −1, 2, and 0 and the sum of squares is 5. For the third row, the differences are 0, −1, 2, and 3 and the sum of squares is 14. The fourth row has a blank/zero entry in the fourth column, so the differences are −1, 2, and −1 and the sum of squares is 6. When we add the sums from each of the four rows, we get $10 + 5 + 14 + 6 = 35$. Finally, we divide by 14 (the number of nonzero elements in A) and take the square root. In this case, $\sqrt{35/14} = 1.581$ is the RMSE.

To decrease RMSE of our objective function incrementally, we can make adjustments to each separate element of U or V matrices. In our running example, we adjust elements of U and V of Fig. 3.9, where all entries are initially set equal to 1, and find values of those entries that give the largest possible improvement in RMSE. Having matrices of Fig. 3.9 as the starting point of UV decomposition, we initially change the value of element u_{11} of matrix U to reduce RMSE as much as possible. Let us denote element u_{11} as variable x, as shown in Fig. 3.10.

Fig. 3.10 The first step of *UV decomposition* method. We set element u_{11} of matrix U to be variable x

$$\begin{bmatrix} x+1 & x+1 & x+1 & x+1 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \end{bmatrix} = \begin{bmatrix} x & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$\hat{A}_{4\times 4} \qquad U_{4\times 2} \qquad V^{\top}_{2\times 4}$$

Please notice in Fig. 3.10 that the only elements of the $\hat{A}$ matrix that change are those in the first row. Thus, when we compare $\hat{A}$ with A, the only change to RMSE comes from the first row.

Fig. 3.11 The only change to RMSE comes from first row

$$\begin{bmatrix} 4 & 1 & 1 & 4 \\ 1 & 4 & 2 & 0 \\ 2 & 1 & 4 & 5 \\ 1 & 4 & 1 & 0 \end{bmatrix} \quad \& \quad \begin{bmatrix} x+1 & x+1 & x+1 & x+1 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \end{bmatrix}$$

$$\hat{A}_{4\times 4} \qquad A_{4\times 4}$$

The contribution to the sum of squares from the first row is as follows:

$$C = (4-(x+1))^2 + (1-(x+1))^2 + (1-(x+1))^2 + (4-(x+1))^2 \iff$$

$$C = (16 - 2(4(x+1)) + (x+1)^2) + (1 - 2(x+1) + (x+1)^2) +$$
$$+ (1 - 2(x+1) + (x+1)^2) + (16 - 2(4(x+1)) + (x+1)^2) \iff$$

$$C = (16 - 8x - 8 + x^2 + 2x + 1) + (1 - 2x - 2 + x^2 + 2x + 1) +$$
$$+ (1 - 2x - 2 + x^2 + 2x + 1) + (16 - 8x - 8 + x^2 + 2x + 1) \iff$$

$$C = (9 - 6x + x^2) + x^2 + x^2 + (9 - 6x + x^2) \iff$$

$$C = (3-x)^2 + x^2 + x^2 + (3-x)^2 = 2((3-x)^2 + x^2) \iff$$

$$C = 2(9 - 6x + 2x^2) \tag{3.32}$$

We want to find the value of x that minimizes the sum. On Eq. 3.32, we take the derivative and set it equal to 0 as follows:

$$\frac{\partial C}{\partial x} = -6 + 4x = 0 \iff$$
$$x = 1.5 \tag{3.33}$$

After computing the value of variable x, initially by predicted values of the first row's elements of $\hat{A}$, which are shown in Fig. 3.9, can be recomputed. The new predicted values of the first row elements of $\hat{A}$ are now shown in Fig. 3.12.

Please notice that for the first row of A and $\hat{A}$ matrices, the sum of the square has been reduced now from 10 to 9. Thus, the total RMSE has been now decreased from 1.581 to 1.558. Let us now assume that we want to change the value of v_{11} element of matrix V. Let us denote the element v_{11} as variable y, as shown in Fig. 3.13.

Fig. 3.12 The results after the first step of the UV decomposition algorithm

$$\begin{bmatrix} 2.5 & 2.5 & 2.5 & 2.5 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \end{bmatrix} = \begin{bmatrix} 1.5 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$
$$\hat{A}_{4\times 4} \qquad U_{4\times 2} \qquad V^{\top}_{2\times 4}$$

Fig. 3.13 The second step of *UV decomposition* method. We set element v_{11} of matrix V to be variable y

$$\begin{bmatrix} 1.5y+1 & 2.5 & 2.5 & 2.5 \\ y+1 & 2 & 2 & 2 \\ y+1 & 2 & 2 & 2 \\ y+1 & 2 & 2 & 2 \end{bmatrix} = \begin{bmatrix} 1.5 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} y & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$
$$\hat{A}_{4\times 4} \qquad U_{4\times 2} \qquad V^{\top}_{2\times 4}$$

In a similar way as Fig. 3.11, we can notice as shown in Fig. 3.14 that only the first column of matrix $\hat{A}$ is affected by y. Thus, when we compare $\hat{A}$ with A, the only change to the RMSE comes from the first column.

Fig. 3.14 The only change to RMSE comes from the first column

$$\underset{\hat{A}_{4\times 4}}{\begin{bmatrix} 4 & 1 & 1 & 4 \\ 1 & 4 & 2 & 0 \\ 2 & 1 & 4 & 5 \\ 1 & 4 & 1 & 0 \end{bmatrix}} \quad \& \quad \underset{A_{4\times 4}}{\begin{bmatrix} 1.5y+1 & 2.5 & 2.5 & 2.5 \\ y+1 & 2 & 2 & 2 \\ y+1 & 2 & 2 & 2 \\ y+1 & 2 & 2 & 2 \end{bmatrix}}$$

The contribution to the sum of squares from the first column is as follows:

$$D = (4 - (1.5y + 1))^2 + (1 - (y + 1))^2 + (2 - (y + 1))^2 + (1 - (y + 1))^2 \iff$$

$$D = (16 - 2(4(1.5y + 1)) + (1.5y + 1)^2 + (1 - 2(y + 1)) + (y + 1)^2 +$$
$$+ (4 - 2(2(y + 1)) + (y + 1)^2 + (1 - 2(y + 1)) + (y + 1)^2 \iff$$

$$D = (16 - 12y - 8 + 2.25y^2 + 3y + 1) + (1 - 2y - 2 + y^2 + 2y + 1) +$$
$$+ (4 - 4y - 4 + y^2 + 2y + 1) + (1 - 2y - 2 + y^2 + 2y + 1) \iff$$

$$D = (9 - 9y + 2.25y^2) + y^2 + (1 - 2y + y^2) + y^2 \iff$$

$$D = 10 - 13y + 5.25y^2 \iff \tag{3.34}$$

As given earlier on Eq. 3.33, we want to find the value of the variable y that minimizes the sum. So on Eq. 3.34 we take the derivative and set it equal to 0, as follows:

$$\frac{\partial D}{\partial y} = -11 + 10.5y = 0 \iff$$
$$y = 1.047 \tag{3.35}$$

After computing the value of variable y, initial by predicted values of the first column's elements of $\hat{A}$, which are shown in Fig. 3.12, can be recomputed. The new predicted values of first row elements of $\hat{A}$ are now shown in Fig. 3.15.

Fig. 3.15 The results after the second step of the UV decomposition algorithm

$$\underset{\hat{A}_{4\times 4}}{\begin{bmatrix} 2.571 & 2.5 & 2.5 & 2.5 \\ 2.047 & 2 & 2 & 2 \\ 2.047 & 2 & 2 & 2 \\ 2.047 & 2 & 2 & 2 \end{bmatrix}} = \underset{U_{4\times 2}}{\begin{bmatrix} 1.5 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}} \times \underset{V^{\top}_{2\times 4}}{\begin{bmatrix} 1.047 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}}$$

By repeating the above steps for every element of U and V matrices, we will try to further minimize RMSE and produce a predicted matrix $\hat{A}$, which will approximate real values of A. Figure 3.16 shows the final predicted matrix A ($\eta = 0.002$, $\beta = 0.02$) for our running example.

$$\underset{\hat{A}}{\begin{bmatrix} 2.56 & 0.82 & 2.36 & 4.08 \\ 1.20 & 3.99 & 1.71 & 0.02 \\ 3.10 & 1.13 & 2.06 & 4.86 \\ 0.74 & 3.89 & 1.31 & -0.71 \end{bmatrix}} = \underset{U}{\begin{bmatrix} 0.67 & 1.67 \\ 1.77 & 0.22 \\ 0.86 & 1.97 \\ 1.68 & -0.52 \end{bmatrix}} \times \underset{V}{\begin{bmatrix} 0.82 & 2.20 & 1.08 & 0.30 \\ 1.21 & -0.39 & 0.97 & 2.33 \end{bmatrix}}$$

Fig. 3.16 The product of matrices U and V, which results in the predicted matrix $\hat{A}$

Please notice that the predicted rating of user U_4 on item I_4 is close to 0. This is as expected because users U_2 and U_4 have similar ratings on items I_1, I_2, and I_3. Thus, user U_4 is predicted to have the same rating with U_2 on item I_4.

3.2.4 The UV Decomposition Algorithm

In Sect. 3.2.3, we described a step-by-step implementation of the UV decomposition method in our running example. In this section, we present (in pseudocode form) the algorithm of UV decomposition [2], as shown in Algorithm 3.1.

The input data in Algorithm 3.1 is the objective function $G = \|A - \hat{A}\|^2$, which should be minimized, the user–item rating matrix A ($A \in \mathbb{R}^{m \times n}$), and the user–user friendship (adjacency) matrix F ($F \in \mathbb{R}^{m \times m}$). Moreover, parameter k controls the latent feature space and the size of U and V matrices, and parameter η controls the step size in the direction of the negative gradient (i.e., the steepest descending direction) of the objective function G.

Based on the parameter k, in lines 1–10 of Algorithm 3.1, we initialize with random values both U and V matrices. Then, as shown in line 11, we initialize function G and compute its partial derivatives ($\frac{\partial G}{\partial U}$ and $\frac{\partial G}{\partial V}$) for U and V, respectively. After the above initializations, the algorithm starts the convergence process of rating prediction in lines 12–16. In particular, in lines 13 and 14, we use the old value of U_i and V_i, respectively, and make a step in the direction of the negative gradient of G. Then, we compute the new gradients $\frac{\partial G}{\partial U}$ and $\frac{\partial G}{\partial V}$ of G, as shown in line 15. This process will be repeatedly executed until the error (i.e., $\|A - \hat{A}\|^2$) between the real and the predicted values of matrices $\hat{A}$ and A ceases to improve or until a maximum number of iterations are reached (line 16). In line 17, we take the product of the computed U and V matrices, which produces the final prediction rating matrix $\hat{A}$.

Algorithm 3.1 The UV decomposition algorithm

Require: The objective function $G = \|A - \hat{A}\|^2$, a user–item rating matrix $A \in \mathbb{R}^{m \times n}$, a user–user (adjacency) matrix $F \in \mathbb{R}^{m \times m}$, parameter k, which controls the size of U and V matrices, and parameter η, which controls the step size in the direction of the negative gradient of the objective function G.
Ensure: An approximation matrix $\hat{A} \in \mathbb{R}^{m \times n}$.

for $i = 1 : m$ **do**
  for $i = 1 : k$ **do**
    Initialize U;
  end for
end for
for $k = 1 : k$ **do**
  for $j = 1 : n$ **do**
    Initialize V;
  end for
end for
Initialize G, $\frac{\partial G}{\partial U}$, $\frac{\partial G}{\partial V}$;
repeat
  $U_i' = U_i - \eta \frac{\partial G}{\partial U}$;
  $V_i' = V_i - \eta \frac{\partial G}{\partial V}$;
  Compute the gradients $\frac{\partial G}{\partial U}$, $\frac{\partial G}{\partial V}$;
until $\|A - \hat{A}\|^2$ ceases to improve **OR** maximum iterations reached
$\hat{A} \leftarrow UV$;
return $\hat{A}$;

3.2.5 *Fusing Friendship into the Objective Function*

In this section, we fuse into the objective function G additional information from the user–user friendship network. To do this, we add a new constraint into the objective function G, which takes into account the friendship among users [3, 6]. The information of the friendship network of users is kept by adjacency matrix F, which is square and symmetric. The symmetry of matrix F is obvious because if a user U_a is a friend of user U_b then user U_b is a friend of user U_a, which means that friendship is reciprocal.

$$\begin{bmatrix} 0 & f_{12} & \cdots & f_{1\,(n-1)} & f_{1n} \\ f_{21} & 0 & \cdots & f_{2\,(n-1)} & f_{2n} \\ \vdots & \cdots & \ddots & \cdots & \vdots \\ f_{(n-1)\,1} & f_{(n-1)\,2} & \cdots & 0 & f_{(n-1)\,n} \\ f_{n\,1} & f_{n\,2} & \cdots & f_{n\,(n-1)} & 0 \end{bmatrix}$$

Friendship matrix $F_{n \times n}$

Both rows and columns of F refer to users, starting with user 1 (U_1) from first row and user n (U_n) for nth row. Respectively, from first column to nth column we have the first user (U_1) and the nth user (U_n). The entries of matrix F are 0 and 1, where 0 means no friendship between users and 1 means that there is friendship. Moreover, intuitively we can notice that the main diagonal of matrix F consists of zeros because a user cannot be a friend of himself.

Next, we add a new constraint in the objective function G of Eq. 3.26. After fusing, the friendship is as follows:

$$G = e_{ij}^2 = (a_{ij} - \sum_{k=1}^{K} u_{ik} v_{kj})^2 + \frac{\beta}{2} \sum_{k=1}^{K} (\|U\|^2 + \|V\|^2) + \frac{\gamma}{2} (\|U - \frac{\sum_{f \in T} U_f}{|T|}\|^2) \tag{3.36}$$

We have added the constraint of friendship which practically influences the prediction of ratings because of friendship among users. This constraint is applied only on the U matrix, the user-feature matrix. Parameter γ is used to control how much friendship influences prediction. In other words, the parameter γ represents the weight in percentage that we want to give on friendship. The values of γ ranged among $[0, 1] \in \mathbb{R}$. If we are trying to predict the rating of user U_a, then the $|T|$ is the set of the users who are friends with U_a. Also, U_f is the value of matrix U that user's U_a friend has given. In simple words, we take the difference among the value of user U_a of matrix U, and the mean of the values on matrix U of all U_a's friends (mean is equal to the quotient of the sum of U_f and total amount of U_a's friends $|T|$). To minimize the objective function G, we have to compute partial derivatives for U and V as follows:

$$\begin{aligned} \frac{\partial G}{\partial U} &= \frac{\partial}{\partial u_{ik}} e_{ij}^2 = 2(a_{ij} - \hat{a}_{ij}) \frac{\partial}{\partial u_{ik}} (a_{ij} - \hat{a}_{ij}) + \beta u_{ik} + \gamma (u_{ik} - \frac{\sum_{f \in T} u_f}{|T|}) \iff \\ \frac{\partial G}{\partial U} &= \frac{\partial}{\partial u_{ik}} e_{ij}^2 = 2(a_{ij} - \hat{a}_{ij})(-v_{kj}) + \beta u_{ik} + \gamma (u_{ik} - \frac{\sum_{f \in T} u_f}{|T|}) \iff \\ \frac{\partial G}{\partial U} &= \frac{\partial}{\partial u_{ik}} e_{ij}^2 = -2 e_{ij} v_{kj} + \beta u_{ik} + \gamma (u_{ik} - \frac{\sum_{f \in T} u_f}{|T|}) \end{aligned} \tag{3.37}$$

$$\wedge$$

$$\begin{aligned} \frac{\partial G}{\partial V} &= \frac{\partial}{\partial v_{kj}} e_{ij}^2 = 2(a_{ij} - \hat{a}_{ij}) \frac{\partial}{\partial v_{kj}} (a_{ij} - \hat{a}_{ij}) + \beta v_{kj} \iff \\ \frac{\partial G}{\partial V} &= \frac{\partial}{\partial v_{kj}} e_{ij}^2 = 2(a_{ij} - \hat{a}_{ij})(-u_{ik}) + \beta v_{kj} \iff \\ \frac{\partial G}{\partial V} &= \frac{\partial}{\partial v_{kj}} e_{ij}^2 = -2 e_{ij} u_{ik} + \beta v_{kj} \end{aligned} \tag{3.38}$$

The above partial derivatives of objective function G are used to make the new update rules as we have done in Sect. 3.2.2. So after adding the new constraint, the new update rules are as follows:

$$u'_{ik} = u_{ik} - \eta \frac{\partial}{\partial u_{ik}} e_{ij}^2 = u_{ik} + \eta [2e_{ij} v_{kj} - \beta u_{ik} - \gamma (u_{ik} - \frac{\sum_{f \in T} u_f}{|T|})] \tag{3.39}$$

$$\wedge$$

$$v'_{kj} = v_{kj} - \eta \frac{\partial}{\partial v_{kj}} e_{ij}^2 = v_{kj} + \eta (2e_{ij} u_{ik} - \beta v_{kj}) \tag{3.40}$$

To check whether our new extended objective function works properly, we have recomputed the predicted matrix $\hat{A}$ in our running example, by exploiting also the user–user friendship network (see Fig. 1.3). As shown in Fig. 1.3, user u_1 is a friend of u_2 and user u_3 is a friend of u_4. The final predicted values of the $\hat{A}$ matrix are shown in Fig. 3.17 ($\gamma = 0.08$, $\eta = 0.002$, $\beta = 0.02$):

$$\underset{\hat{A}}{\begin{bmatrix} 2.54 & 0.82 & 2.32 & 4.04 \\ 1.16 & 3.94 & 1.67 & 0.05 \\ 3.07 & 1.14 & 2.82 & 4.80 \\ 0.80 & 3.90 & 1.31 & -0.60 \end{bmatrix}} = \underset{U}{\begin{bmatrix} 1.12 & 0.47 \\ -0.18 & 1.22 \\ 1.33 & 0.61 \\ -0.37 & 1.15 \end{bmatrix}} \times \underset{V}{\begin{bmatrix} 1.74 & -0.58 & 1.39 & 3.34 \\ 1.21 & 3.12 & 1.57 & 0.56 \end{bmatrix}}$$

Fig. 3.17 The product of matrices U and V, which results in the predicted matrix $\hat{A}$, which also takes into consideration the information of the users' friendship network

3.2.6 Inserting New Data in the Initial Matrix

In this section, we will see how we can add new data (new users-rows or new items-columns) in the initial matrix A. The process of new data insertion is important for online collaborative filtering. This method has been developed for real-time data analysis in an online context. The classic method of UV decomposition has been used for static data analysis and pattern recognition. The aim of the online version of UV decomposition is to perform a rapid analysis so that recommendations can be produced in real time and adapt as new data are inserted into the recommendation engine.

We begin our study with an initial factorization of matrix A at time point t, where we have an $m \times n$ data matrix A. For simplicity, we claim that Eq. 3.18 is applicable. At time point $t+1$, we add the additional data of matrix M into the initial matrix A resulting in $\tilde{A}$:

$$\tilde{A} = \begin{pmatrix} A \\ M \end{pmatrix} \tag{3.41}$$

The additional data M are rows and/or columns that are added to our initial matrix A. In this way, we produce the $\tilde{A}$ matrix with more data.

So now the problem of online version of UV is how to integrate U and V into $\tilde{U}$ and $\tilde{V}$ so that:

$$\tilde{A} = \tilde{U}\tilde{V} \tag{3.42}$$

The following theorem makes it possible to design an online version of UV decomposition.

Theorem 1 (Full-Rank Decomposition Theorem) *If $A = UV$ and $A = U'V'$ are both full-rank decompositions, then there exists one invertible matrix P satisfying $U = U'P$ and $V = P^{-1}V'$.*

Proof With the condition $UV = U'V'$, by multiplying $V^\top$ on both sides we have $UVV^\top = U'V'V^\top$. From full-rank condition, we get $U = U'V'(VV^\top)^{-1} = U'P$, where $P = V'(VV^\top)^{-1}$. As the same, we can get $V = (U^\top U)^{-1}U^\top U'V' = QV'$. It is easy to validate $PQ = QP = I$ where I is the identity matrix. Therefore, $Q = P^{-1}$.

Considering the *full-rank decomposition theorem*, we have

$$\tilde{A} = \begin{pmatrix} A \\ M \end{pmatrix} = \tilde{U}\tilde{V} = \begin{pmatrix} \tilde{U}_1 \\ \tilde{U}_2 \end{pmatrix} \tilde{V} \tag{3.43}$$

where $\tilde{U}_1$ and $\tilde{U}_2$ are blocks of $\tilde{U}$ corresponding to A and M, respectively. From Eq. 3.43 we have:

$$A = \tilde{U}_1\tilde{V} \tag{3.44}$$

For Eqs. 3.18 and 3.44, using the *full-rank decomposition theorem* which is described above, we will get

$$\tilde{U}_1 = UP \tag{3.45}$$

$$\wedge$$

$$\tilde{V} = P^{-1}V \tag{3.46}$$

Thus, the factorization problem is converted to:

$$M = \tilde{U}_2\tilde{V} \quad s.t. : \tilde{V} = P^{-1}V \tag{3.47}$$

To find a solution to Eq. 3.47, we consider factorization of the new data matrix by replacing A by V:

$$\begin{pmatrix} V \\ M \end{pmatrix} = U^* V^* = \begin{pmatrix} U_1^* \\ U_2^* \end{pmatrix} V^* \tag{3.48}$$

By solving this problem we obtain

$$V = U_1^* V^* \tag{3.49}$$

$$\wedge$$

$$M = U_2^* V^* \tag{3.50}$$

Assume that $\tilde{U_1^*}$ is invertible, so from Eq. 3.49 we take

$$V^* = U_1^{*-1} V \tag{3.51}$$

Now, we get the solution to Eq. 3.47 by setting $\tilde{V} = V^*$, $P = U_1^{*-1}$, and $\tilde{U_2} = U_2^*$. Based on the previous factorization result $A = UV = UU_1^* V^*$, we have

$$\tilde{A} = \begin{pmatrix} UU_1^* \\ U_2^* \end{pmatrix} \tilde{V} = \tilde{U}\tilde{V} \tag{3.52}$$

Finally, we update factor rules:

$$\tilde{U} = \begin{pmatrix} UU_1^* \\ U_2^* \end{pmatrix} \tag{3.53}$$

$$\wedge$$

$$\tilde{V} = U_1^{*-1} V \tag{3.54}$$

Since the solution to Eq. 3.52 is solved by minimizing a target function which is not convex, the solution to Eq. 3.43 is an approximation rather an exact solution. However, the following analysis shows that the approximate solution is reasonable.

As we have seen, the UV decomposition algorithm tries to find a set of bases to represent input data by linear combination. When new data arrive, the bases need to be updated to represent the new data. Since old bases can be used to represent old data, we can update bases using previous bases and new data instead of using all the data. This is the main philosophy behind the online version of UV decomposition and the difference between online and classic UV decomposition. To adjust contributions of old factors, we modify the online version of UV decomposition by introducing a weighting schema. That is, we can use SV to replace V in Eq. 3.52. The new matrix S is a diagonal matrix with S_{ii} representing the weight of factor v_i. Then, the relation between old and new factors is as follows:

$$V^* = {U_1}^{-1} SV \tag{3.55}$$

and the update rules become:

$$\tilde{U} = \begin{pmatrix} U S^{-1} U_1^* \\ U_2^* \end{pmatrix} \tag{3.56}$$

$$\wedge$$

$$\tilde{V} = U_1^{*-1} SV \tag{3.57}$$

When the data are sparse or incomplete in terms of their distribution, the UV-decomposition method may find latent factors incorrectly. This problem is referred to as the *partial-data* problem which is related to the unique solution problem of UV decomposition.

Theorem 2 *Suppose that we are given a nonnegative factorization $A = UV$, where U and V satisfy $U = P_1 \binom{\Delta_1}{U_1}$, $V = (\Delta_2 V_1) P_2$ and where P_1 and P_2 are permutation matrices, while Δ_1 and Δ_2 are diagonal matrices. The factorization is unique under permutation (apart from a scaling factor).*

Theorem 2 clarifies the condition for unique UV decomposition. But Theorem 2 requires the latent factors to have distinct features from each other and the data distribution should be complete. To make the solution unique when the data are incomplete, more strict requirements are needed for factors.

Theorem 3 *If we restrict V to satisfy $\boldsymbol{v}_i \boldsymbol{v}_j = 0 \quad for \quad i \neq j$ ($\boldsymbol{v}_i, \boldsymbol{v}_j$ are the ith and jth rows of V), then the nonnegative factorization $A = UV$ is unique under permutation (apart from a scaling factor).*

Then, with Theorem 3 we introduce orthogonal constraints on the online version of UV decomposition to tackle the partial-data problem. Theorem 3 requires the factors to be orthogonal; thus, the decomposition problem is converted to a minimization problem:

$$min J = \frac{1}{2} \|A - UV\|_F^2 \quad s.t. : U \geq 0, V \geq 0, \mathbf{v}_i \mathbf{v}_j = 0, i \neq j \tag{3.58}$$

where $\|A - UV\|_F^2 = \sum_{ij} (\mathbf{a} - \mathbf{uv})^2$. By introducing a regularization coefficient for orthogonality constraint, the minimization problem is further converted to the following problem:

$$J = \frac{1}{2} \|A - UV\|_F^2 + \mathbf{a} \Gamma V V^\top \tag{3.59}$$

where γ is a symmetry matrix with diagonal elements equal to zero and a is a positive integer. We can obtain a solution to Eq. 3.58 by the following iterative formulas:

$$u_{ij} \leftarrow u_{ij} \frac{(AV^\top)_{ij}}{(UVV^\top)_{ij}} \tag{3.60}$$

$$\wedge$$

$$v_{ij} \leftarrow v_{ij} \frac{(U^\top A)_{ij}}{(U^\top UV + a\Gamma V)_{ij}} \tag{3.61}$$

References

1. Berry, M., Dumais, S., O'Brien, G.: Using linear algebra for intelligent information retrieval. SIAM Rev. **37**(4), 573–595 (1994)
2. Bhargava, P., Phan, T., Zhou, J., Lee, J.: Who, what, when, and where: multi-dimensional collaborative recommendations using tensor factorization on sparse user-generated data. In: Proceedings of the 24th International Conference on World Wide Web, WWW'15, Republic and Canton of Geneva, Switzerland, pp. 130–140. International World Wide Web Conferences Steering Committee (2015)
3. Forsati, R., Mahdavi, M., Shamsfard, M., Sarwat, M.: Matrix factorization with explicit trust and distrust side information for improved social recommendation. ACM Trans. Inf. Syst. **32**(4), 17:1–17:38 (2014)
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. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Application of dimensionality reduction in recommender system-a case study. In: ACM WebKDD Workshop (2000)
6. Yuan, Q., Chen, L., Zhao, S.: Factorization vs. regularization: fusing heterogeneous social relationships in top-n recommendation (2011)

Chapter 4
Experimental Evaluation on Matrix Decomposition Methods

Abstract In this chapter, we study the performance of described SVD and UV decomposition algorithms, against an improved version of the original item based CF algorithm combined with SVD. For the UV decomposition method, we will present the appropriate tuning of parameters of its objective function to have an idea of how we can get optimized values of its parameters. We will also answer the question if these values are generally accepted or they should be different for each data set. The metrics we will use are root-mean-square error (RMSE), precision, and recall. The size of a training set is fixed at 75 %, and we perform a fourfold cross-validation.

Keywords Experiments · SVD decomposition · UV decomposition

4.1 Data Sets

We perform experiments with two real data sets that have been used as benchmarks in prior work.

The first data set has been extracted from the www.epinions.com website (http://www.epinions.com). Epinions is a website that contains reviews of users on items such as electronics, movies, books, music, etc. Users also build their web of trust within the Epinions community. This web of trust is a list of trusted and distrusted users. Notice that trusted users' reviews are promoted, whereas distrusted users' reviews are less likely encountered. A review contains a rating between 1 and 5 and a free text message. Reviews can be commented and/or rated. This data set contains 131.828 users who have rated 841.372 items.

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_4

The second data set has been extracted from the following GeoSocialRec website (http://delab.csd.auth.gr/geosocialrec), which is a location-based social network. We have chosen a small data set to test our algorithm in a case of severe sparseness and lack of user's data. The data set consists of 212 users who rated 649 locations. GeoSocialRec offers to its users the ability to connect to each other, to declare their position with a check-in, and to rate the places they have visited.

4.2 Sensitivity Analysis of the UV Decomposition Algorithm

4.2.1 Tuning of the k Latent Feature Space

In this section, we will examine how parameter k affects the effectiveness and performance of our algorithm. Recall that parameter k controls the size of U and V matrices which store user-latent features and item-latent features, respectively. Thus, a small number of k latent features mean that we keep both matrices thin, whereas the required storage space is small. To tune parameter k, we fix the number of algorithm's iteration at 5.000, which is the number of times that our algorithm is applied on data to predict ratings. The procedure goes like this: On the first algorithm's iteration, we predict missing ratings and store them in the training set. On the second algorithm's iteration, we repredict ratings using the information stored in the previous step and so on.

For the Epinions data set, as shown in Fig. 4.1, RMSE decreases with increasing values of parameter k. The best RMSE is attained when parameter k is equal to 16 latent features.

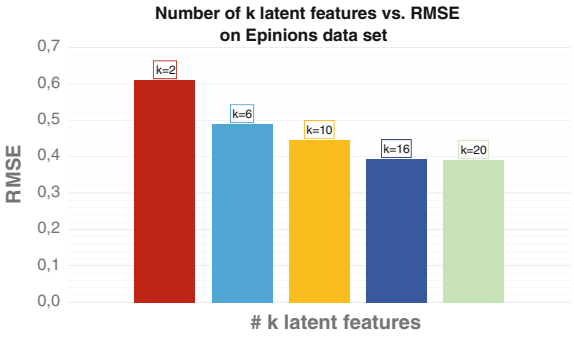

Fig. 4.1 RMSE versus different number of k latent features on the Epinions data set

In contrast, for the GeoSocialRec data set, as shown in Fig. 4.2, the best RMSE is attained for $k = 2$. That is, for different data sets, there can be no rule for selecting the optimal value of parameter k, as it depends on the size of the data set, the data sparseness, and probably other factors, which may be particular for each data set.

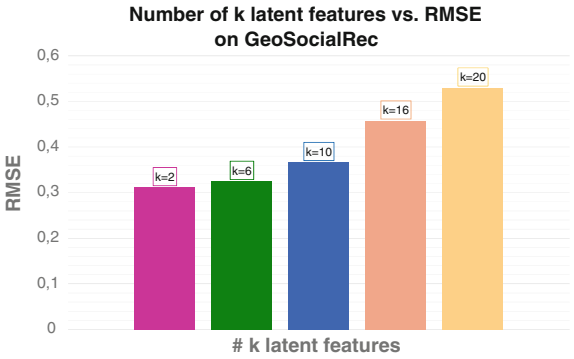

Fig. 4.2 RMSE versus different number of k latent features on the GeoSocialRec data set

4.2.2 Tuning of Parameter β

Parameter β is responsible to control the magnitudes of item-latent features and user-latent features of U and V matrices, respectively. Parameter β overcomes the problem of overfitting. For both data sets, as shown in Figs. 4.3 and 4.4, RMSE drops slightly with decreasing values of the parameter β. In particular, with smaller values of β, we get an improvement of 20 % in the Epinions data set and only 2 % in GeoSocialRec data set. As expected, data overfitting does not exist in the GeoSocialRec data set, since there is not enough training data to bias our algorithm. In contrast, parameter β is important in the case of the Epinions data set, where it improves drastically the performance of our algorithm.

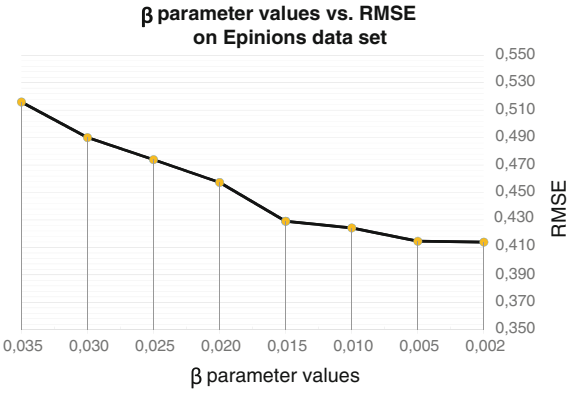

Fig. 4.3 RMSE versus different values of parameter β on the Epinions data set

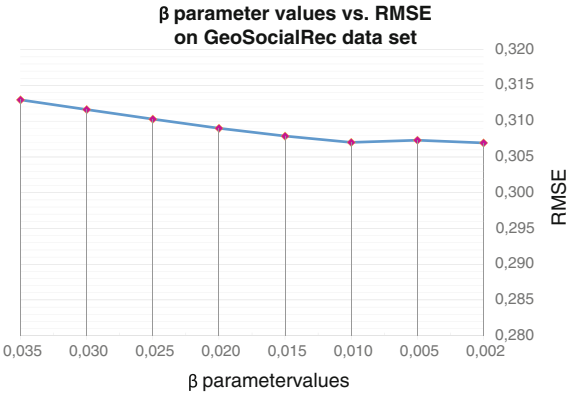

Fig. 4.4 RMSE versus different values of parameter β on the GeoSocialRec data set

4.2.3 Optimizing Algorithm's Parameters

In this section, we present the algorithm's performance with default values versus the performance after tuning the parameters of the objective function.

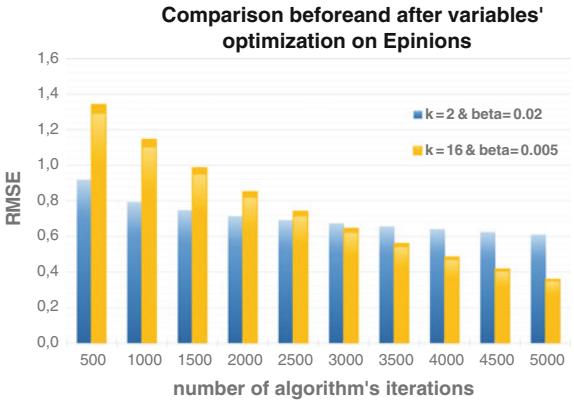

Fig. 4.5 RMSE results using default versus optimized parameters of the objective function on the GeoSocialRec

For the Epinions data set, as shown in Fig. 4.5, blue bars present RMSE levels calculated with the initial algorithm's parameter values, whereas yellow bars present RMSE levels after optimizing the parameters of the algorithm's objective function. As expected, as algorithm's iterations for rating predictions are increased, RMSE of optimized parameters outperforms the performance of initial values. Please notice that the improvement of optimized parameters over initial variables is quite significant

for this data set. The main reason is that there is enough information in the training data so that the UV decomposition algorithm can make accurate "generalizations" for the test data.

For the GeoSocialRec data set, as shown in Fig. 4.6, improvement of RMSE on GeoSocialRec data set is not significant. The main reason is that this data set is very small and there is not enough information in the training set.

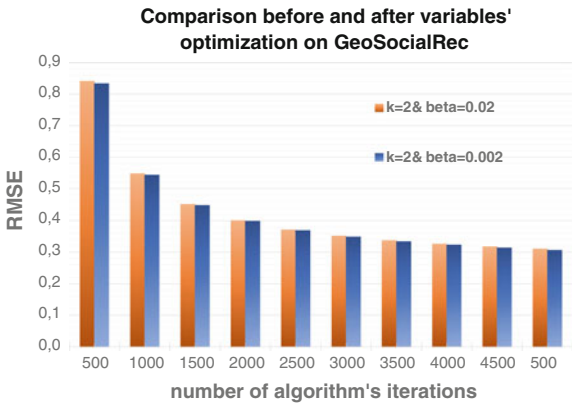

Fig. 4.6 RMSE results using default versus optimized parameters of the objective function on the GeoSocialRec

4.3 Comparison to Other Decomposition Methods

In this section, we compare the UV decomposition algorithm (with parameters that attained the best performance in previous experiments) against the following methods:

- CUR-decomposition algorithm, which confronts the problem of high density in the factorized matrices (a problem that is faced mainly when using the SVD decomposition method) [1, 3, 4]. This algorithm is denoted as CUR.
- item-based CF combined with SVD [6]. Item-based CF is an improved version [2] of the well-known item-based CF algorithm that weights similarities by the number of common ratings among items. This variation of item-based CF weights the similarity sim between two items with a parameter γ, as follows: $\frac{\max(c,\gamma)}{\gamma} \cdot sim$, where c is the number of co-rated users. The best value of parameter γ is fixed at 4 and 2 for the Epinions and the GeoSocialRec data set, respectively. This algorithm is denoted as item-based SVD.

We will compare all three algorithms in terms of precision, recall, and RMSE measures. Please notice that RMSE works well for measuring how accurately an

algorithm predicts the rating of a randomly selected item, but may fail to evaluate whether an algorithm will provide accurate item recommendations [5]. Therefore, precision–recall evaluation measures, in addition to RMSE, are able to better measure the quality of recommendations. Moreover, we will use precision–recall diagrams because they can reveal the robustness of each algorithm in attaining high recall with minimal losses in terms of precision. We examine the top-N ranked item list, which is recommended to a target user, starting from the top item. In this case, recall and precision vary as we proceed with the examination of the top-N list of recommended items.

For the Epinions data set, in Fig. 4.7a, we plot a precision versus a recall curve for all three algorithms. As expected, all algorithms' precision falls as N increases. In contrast, as N increases, recall for all algorithms increases as well. The UV decomposition algorithm attains the best results.

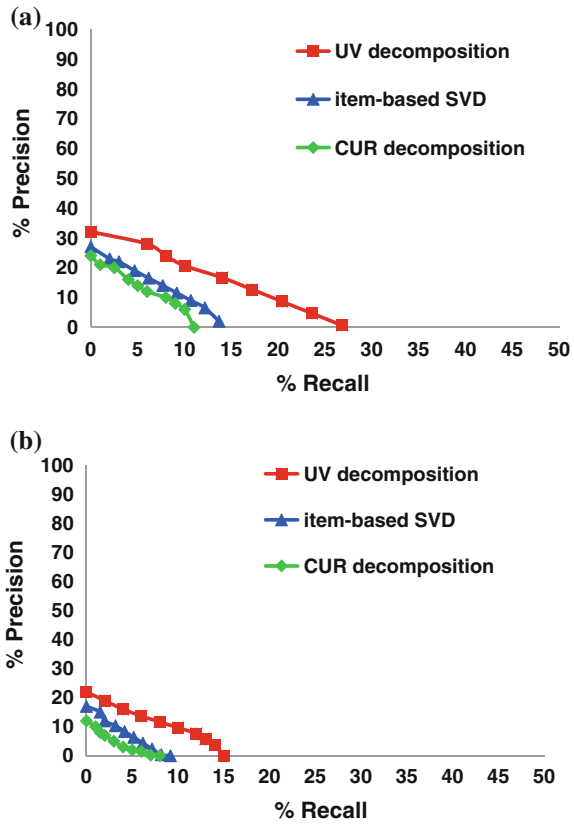

Fig. 4.7 Accuracy performance of algorithms in terms of precision–recall for the **a** Epinions and **b** GeoSocialRec data sets

For the GeoSocialRec data set, in Fig. 4.7b we also plot a precision versus recall diagram. The UV decomposition algorithm again outperforms the item-based SVD and CUR algorithms. Notice that the accuracy performance of all algorithms for the GeoSocialRec data set is lower than those for the Epinions data set. The reason is possibly because the latter has more ratings per user and can be considered a more dense data set.

Next, we measured RMSE for all three examined algorithms on the Epinions and GeoSocialRec data sets. The results are summarized in Table 4.1. Again, UV decomposition clearly outperforms the item-based SVD and CUR algorithms in terms of RMSE. As shown, UV decomposition attains the lowest RMSE values and item-based SVD is the second best algorithm in both data sets.

Table 4.1 RMSE values for all three algorithms on two real data sets

Algorithm	Epinions data set	GeoSocialRec data set
UV decomposition	0.38	0.32
Item-based SVD	0.76	0.98
CUR decomposition	0.79	1.08

A smaller value means a better performance

In conclusion, we have to notice that good results of RMSE may not fully characterize users' experience in web-based recommender systems (Amazon, eBay, etc.), which propose a top-N ranked list of items to a user. The basic reason is that an error of size ϵ has the same impact on RMSE regardless of where that error places the item in a top-N ranking. In contrast, the results of precision–recall can better characterize the users' experience in the aforementioned systems.

References

1. Drineas, P., Kannan, R., Mahoney, M.W.: Fast monte carlo algorithms for matrices III: computing a compressed approximate matrix decomposition. SIAM J. Comput. **36**(1), 184–206 (2006)
2. Herlocker, J., Konstan, J., Riedl, J.: An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. Inf. Retrieval **5**(4), 287–310 (2002)
3. Mahoney, M.W., Drineas, P.: Cur matrix decompositions for improved data analysis. Proc. Natl. Acad. Sci. **106**(3), 697–702 (2009)
4. Mahoney, M.W., Maggioni, M., Drineas, P.: Tensor-cur decompositions for tensor-based data. SIAM J. Matrix Anal. Appl. **30**(3), 957–987 (2008)
5. McLauglin, R., Herlocher, J.: A collaborative filtering algorithm and evaluation metric that accurately model the user experience. In: Proceedings of ACM SIGIR Conference, pp. 329–336 (2004)
6. Symeonidis, P., Nanopoulos, A., Papadopoulos, A., Manolopoulos, Y.: Collaborative recommender systems: combining effectiveness and efficiency. Expert Syst. Appl. **34**(4), 2995–3013 (2008)

Part II
Tensor Factorization Techniques

Chapter 5
Related Work on Tensor Factorization

Abstract In this chapter, we provide a preliminary knowledge overview of tensors. Moreover, we provide the related work on tensor decomposition methods. The first method that is discussed is the Tucker Decomposition (TD) method, which is the underlying tensor factorization model of Higher Order Singular Value Decomposition. TD decomposes a tensor into a set of matrices and one small core tensor. The second one is the PARAFAC method (PARAllel FACtor analysis), which is the same as the TD method with the restriction that the core tensor should be diagonal. The third method is the Pairwise Interaction Tensor Factorization method, which is a special case of the TD method with linear runtime both for learning and prediction. The last method that is analyzed is the low-order tensor decomposition (LOTD) method. This method has low functional complexity, is uniquely capable of enhancing statistics, and avoids overfitting compared with traditional tensor decompositions such as TD and PARAFAC.

Keywords Tensor decomposition

5.1 Preliminary Knowledge of Tensors

Formally, a *tensor* is a multidimensional 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 higher order singular value decomposition [10] generalizes the singular value decomposition (SVD) computation to tensors. To apply the higher order singular value decomposition (HOSVD) technique on a third-order tensor $\mathcal{A}$, three *matrix unfolding*[1] operations are defined as follows [10]:

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

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_5

$$A_1 \in \mathbb{R}^{I_1 \times (I_2 I_3)}, \qquad A_2 \in \mathbb{R}^{I_2 \times (I_1 I_3)}, \qquad A_3 \in \mathbb{R}^{(I_1 I_2) \times I_3} \tag{5.1}$$

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 Fig. 5.1.

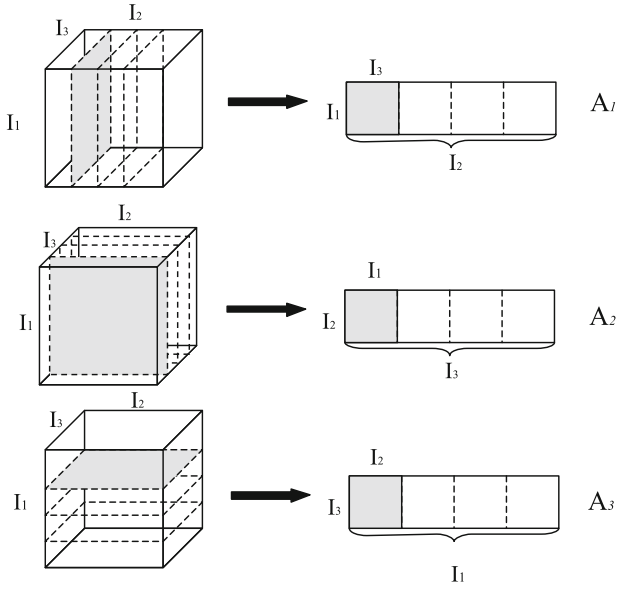

Fig. 5.1 Visualization of the three unfoldings of a third-order tensor

In the following, we will present an example of tensor decomposition adopted from [10]:

Example 1 Define a tensor $\mathcal{A} \in \mathbb{R}^{3\times 2\times 3}$ 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 Fig. 5.2.

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:

$$(\mathcal{A} \times_n U)_{i_1 i_2 \ldots i_{n-1} j_n i_{n+1} \ldots i_N} = \sum_{i_n} a_{i_1 i_2 \ldots i_{n-1} i_n i_{n+1} \ldots i_N} u_{j_n, i_n} \tag{5.2}$$

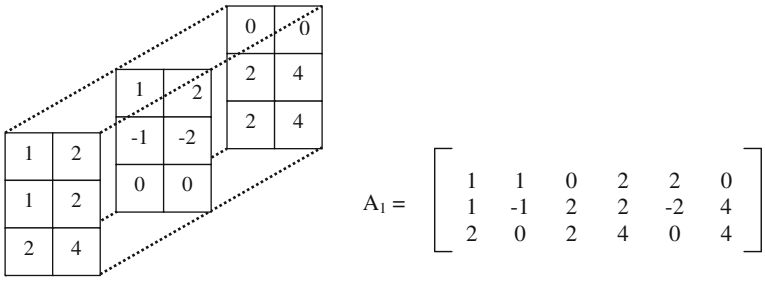

Fig. 5.2 Visualization of tensor $\mathcal{A} \in \mathbb{R}^{3\times2\times3}$ and its mode-1 matrix unfolding

Since we focus on third-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., second-order tensor) can be rewritten as follows [10]:

$$F = S \times_1 U^{(1)} \times_2 U^{(2)} \tag{5.3}$$

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

i. pseudodiagonality: $S = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_{\min\{I_1, I_2\}})$ and
ii. ordering: $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_{\min\{I_1, I_2\}} \geq 0$.

By extending this form of SVD, HOSVD of a third-order tensor $\mathcal{A}$ can be written as follows [10]:

$$\mathcal{A} = S \times_1 U^{(1)} \times_2 U^{(2)} \times_3 U^{(3)} \tag{5.4}$$

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 the core tensor and has the property of "all-orthogonality."[3] This decomposition also refers to a general factorization model known as Tucker decomposition [20].

[2] 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, $U^T U = I_n$.

[3] 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 products of corresponding entries vanishes); at the same time, different "frontal" matrices (i_2 fixed) and different "vertical" matrices (i_3 fixed) should be mutually orthogonal as well. For more information, see [10].

In the following, we will discuss several factorization models that have been proposed for tag recommendation. We investigate their model assumptions, complexity, and their relations among each other.

5.2 Tucker Decomposition and HOSVD

The Tucker decomposition (TD) was first introduced by Tucker [20] in 1963. The Tucker I decomposition method is an important variation of the Tucker decomposition, which is later known as HOSVD [10]. HOSVD decomposes a tensor into a set of matrices and one small core tensor. In this section, we elaborate on how HOSVD can be employed for tensor factorization in social tagging systems (STSs).

The ternary relation of users, items, and tags in STSs 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 $\mathcal{A}$. The idea of computing low-rank tensor approximations has already been used for many different purposes [3, 9, 10, 17, 18, 22]. The basic idea is to cast the recommendation problem as a third-order tensor completion problem by completing the nonobserved 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 nonempty 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 nonempty set $T_{u,i} := \{t \in T \mid (u, i, t) \in Y\}$.

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

$$\hat{\mathcal{A}} := \hat{C} \times_u \hat{U} \times_i \hat{I} \times_t \hat{T} \tag{5.5}$$

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

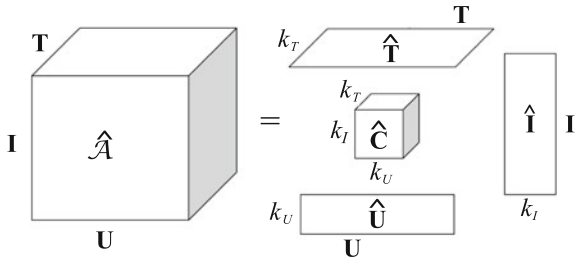

Fig. 5.3 Tensor decomposition in STS. Figure adapted from [15]

latent factors. The model parameters to be optimized are represented by the quadruple $\hat{\theta} := (\hat{C}, \hat{U}, \hat{I}, \hat{T})$ (see Fig. 5.3).

The basic idea of the HOSVD algorithm is to minimize an elementwise 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:

$$\hat{a}(u,i,t) := \sum_{\tilde{u}=1}^{k_U} \sum_{\tilde{i}=1}^{k_I} \sum_{\tilde{t}=1}^{k_T} \hat{c}_{\tilde{u},\tilde{i},\tilde{t}} \cdot \hat{u}_{u,\tilde{u}} \cdot \hat{i}_{i,\tilde{i}} \cdot \hat{t}_{t,\tilde{t}} \tag{5.6}$$

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., $\hat{t}_{t,\tilde{t}}$).

Please notice that there are incremental solutions to update the tensor, as more data are accumulated to the system. However, please notice that the reason for the cubic complexity (i.e., $O(k^3)$ with $k := \min(k_U, k_I, k_T)$) of HOSVD is the core tensor.

5.3 AlsHOSVD

The reconstructed tensor of the previous subsection (also known as truncated HOSVD) is not optimal, but is a good starting point for an iterative alternating least squares (ALS) algorithm to best fit the reconstructed tensor to the original one [8]. The basic idea of the AlsHOSVD algorithm tries to minimize theerror between the

initial and the predicted values of the tensor. The pseudocode of the approach is depicted in Algorithm 5.1.

Algorithm 5.1 AlsHOSVD

Require: The initial tensor $\mathcal{A}$ with user, tag, and item dimensions.
Ensure: The approximate tensor $\hat{\mathcal{A}}$ with k_U, k_I and k_T left leading eigenvectors of each dimension, respectively.

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

As shown in line 8 of Algorithm 5.1, AlsHOSVD minimizes an objective function that computes the error among real and predicted values of original and approximate tensors. This is done cyclically until our objective function ceases to fit to the original values or the maximum number of user-defined iterations is reached. Please notice that values of leading left singular vectors in all three modes (lines 5–7) increased gradually in each repetition.

5.4 Parallel Factor Analysis (PARAFAC)

The PARAFAC [6] model a.k.a. canonical decomposition [2] (CANDECOMP) reduces the complexity of the TD model by assuming only a diagonal core tensor.

$$c_{\tilde{u},\tilde{i},\tilde{t}} \stackrel{!}{=} \begin{cases} 1, & \text{if } \tilde{u} = \tilde{i} = \tilde{t} \\ 0, & \text{else} \end{cases} \tag{5.7}$$

which allows to rewrite the model equation:

$$\hat{a}_{u,i,t} = \sum_{f=1}^{k} \hat{u}_{u,f} \cdot \hat{i}_{i,f} \cdot \hat{t}_{t,f}, \text{ for } u = 1, \ldots, U, i = 1, \ldots, I, t = 1, \ldots, T \tag{5.8}$$

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

$$\hat{U} \in \mathbb{R}^{|U| \times k}, \quad \hat{I} \in \mathbb{R}^{|I| \times k}, \quad \hat{T} \in \mathbb{R}^{|T| \times k} \tag{5.9}$$

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

A graphical representation of TD and PARAFAC is shown in Fig. 5.4. It is seen that any PARAFAC model can be expressed by a TD model (with diagonal core tensor).

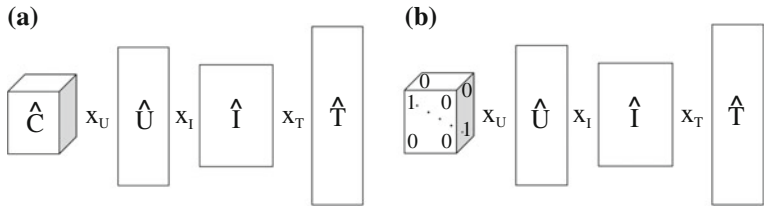

Fig. 5.4 Relationship between Tucker decomposition (TD) and parallel factor analysis (PARAFAC)

Let $\mathcal{M}$ be the set of models that can be represented by a model class. In [14], it is shown that for tag recommendation

$$\mathcal{M}^{\text{PARAFAC}} \subset \mathcal{M}^{\text{TD}} \tag{5.10}$$

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 [14, 16] 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 a higher prediction quality if the restriction is in line with true parameters.

5.5 Pairwise Interaction Tensor Factorization (PITF)

Rendle and Schmidt-Thieme [13] proposed the (PITF) model, which is a special case of the TD model with a linear runtime both for learning and prediction. PITF explicitly models pairwise interactions between users, items, and tags. Whereas TD and PARAFAC directly express a ternary relation, the idea of PITF is to model pairwise interactions instead. The motivation is that observations are typically very limited and sparse in tag recommendation data, and thus it is often easier to estimate pairwise interactions than ternary ones. This assumption is reflected in the model equation of PITF which reads:

$$\hat{a}_{u,r,t} = \sum_{f}^{k} \hat{u}_{u,f} \cdot \hat{t}^{U}_{t,f} + \sum_{f}^{k} \hat{i}_{i,f} \cdot \hat{t}^{I}_{t,f} \tag{5.11}$$

with model parameters $\hat{\theta}$

$$\hat{U} \in \mathbb{I}^{|U| \times k}, \quad \hat{I} \in \mathbb{I}^{|I| \times k}, \quad \hat{T^U} \in \mathbb{I}^{|T| \times k}, \quad \hat{T^I} \in \mathbb{I}^{|T| \times k} \tag{5.12}$$

Note that in contrast to PARAFAC, there are two factor matrices for tags: one (T^U) for the interaction of tags with users and a second one (T^I) for the interaction of tags with items.

5.6 PCLAF and RPCLAF Algorithms

In this section, we elaborate on how tensor decomposition techniques can be employed in location-based social networks (LBSNs). The ternary relation of users, locations, and activities in LBSNs can be represented as a third-order tensor.

Zheng et al. [23] introduced a personalized recommendation algorithm for LBSNs, which performs personalized collaborative location and activity filtering (PCLAF). PCLAF treats each user differently and uses a collective tensor and matrix factorization to provide personalized recommendations. As shown in Fig. 5.5, the novelty of PCLAF lies in the utilization of a user–location–activity tensor along with user–user, user–location, location–features, and activity–activity matrices.

As also shown in Fig. 5.5, to fill missing entries in the tensor $\mathcal{A}$, PCLAF decomposes $\mathcal{A}$ w.r.t. each tensor dimension (i.e., user, location, activity). Then, PCLAF forces latent factors to be shared with additional matrices to utilize their information. After such latent factors are obtained, PCLAF reconstructs an approximation tensor $\widehat{\mathcal{A}}$ by filling all missing entries. Notice that PCLAF uses a PARAFAC-style regularized tensor decomposition framework to integrate the tensor with additional matrices. In particular, Zheng et al. [23] construct a third-order tensor $\mathcal{A}$, which captures relations among users X, locations Y, activities Z, and location features U.

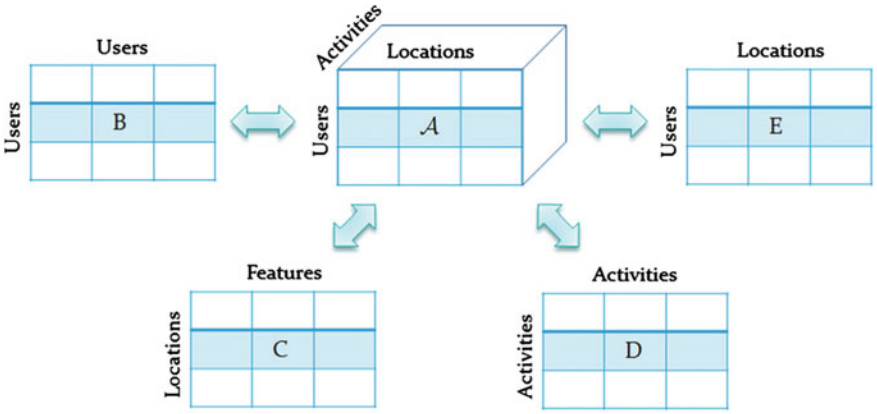

Fig. 5.5 Visual representation of a user–location–activity tensor along with user–user, user–location, location–features, and activity–activity matrices adapted from [23]

They initially decompose tensor $\mathcal{A}$ to three low-dimensional representations with respect to each tensor entity (i.e., users, locations, and activities). Then, they reconstruct the tensor trying to fill all its missing entries. To do so, they exploit additional information from user–user, location–feature, activity–activity, and location–activity matrices. They want to minimize the error between real and predicted values of the reconstructed tensor as shown in the following objective function Eq. 5.13:

$$\begin{aligned}\mathcal{L}(X, Y, Z, U) = &\frac{1}{2}\|A - [X, Y, Z]\|^2 + \frac{\lambda_1}{2} tr(X^T L_B X) \\ &+ \frac{\lambda_2}{2}\|C - YU^T\|^2 + \frac{\lambda_3}{2} tr(Z^T L_D Z) \\ &+ \frac{\lambda_4}{2}\|E - XY^T\|^2 + \frac{\lambda_5}{2}(\|X\|^2 + \|Y\|^2 + \|Z\|^2 + \|U\|^2)\end{aligned} \tag{5.13}$$

where B denotes the user–user matrix, C is the location–feature matrix, D is the activity–activity matrix, and E is the location–activity matrix. L_B and L_D are Laplacian matrices of B and D, respectively (i.e., $L_B = Q - B$ and $L_D = Q - D$, where Q is a diagonal matrix). tr denoted as the trace of a matrix. Finally, λ_i are model parameters.

In addition, Zheng et al. [24] proposed the ranking-based personalized collaborative location and activity filtering (RPCLAF). RPCLAF takes a direct way to solve the recommendation problem using a ranking loss objective function. That is, instead of minimizing the prediction error between the real and predicted user preference for an activity in a location, the RPCLAF method formulates the user's location–activity pairwise preferences by Eq. 5.14:

$$\theta_{u,l,a,a'} := \begin{cases} +1, & \textbf{if } \mathcal{A}_{u,l,a} > \mathcal{A}_{u,l,a'} \mid (u,l,a) \in I_i \wedge (u,l,a') \notin I_i; \\ 0, & \textbf{if } \mathcal{A}_{u,l,a} = \mathcal{A}_{u,l,a'} \mid (u,l,a) \in I_i \wedge (u,l,a') \in I_i; \\ -1, & \textbf{if } \mathcal{A}_{u,l,a} < \mathcal{A}_{u,l,a'} \mid (u,l,a) \notin I_i \wedge (u,l,a') \in I_i; \\ ?, & \textbf{if } (u,l,a) \notin I_i \vee (u,l,a') \notin I_i \end{cases} \tag{5.14}$$

where I_i denotes location–activity pairwise preferences of user i in tensor $\mathcal{A}$, $\mathcal{A}_{u,l,a}$ denotes the preference of user u on the activity a that she performed in location l, whereas $\mathcal{A}_{u,l,a'}$ denotes the preference for user u on the activity k' that she performed in location l. Based on Eq. 5.14, RPCLAF distinguishes between positive and negative location–activity pairwise preferences and missing values to learn a personalized ranking of activities/locations. The idea is that positive $(+1)$ and negative examples (-1) are only generated from observed location–activity pairwise preferences. Observed location–activity pairwise preferences are interpreted as positive feedback $(+1)$, whereas nonobserved location–activity pairwise preferences are marked as negative (-1) feedback. All other entries are assumed to be either missing (?) or zero values.

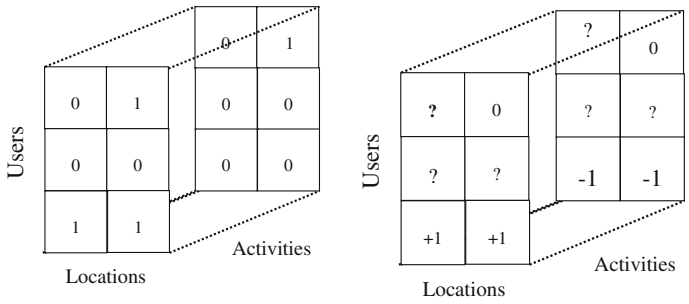

Fig. 5.6 *Left* Tensor's data representation of the HOSVD algorithm (*left*) and the RPCLAF algorithms (*right*)

To give a more clear view of tensor representation based on ranking, in Fig. 5.6, we compare tensor representation of the HOSVD [19] algorithm, with the tensor representation of RPCLAF.

The left-hand side of Fig. 5.6 shows the tensor representation of the HOSVD algorithm [19], where the positive feedback is interpreted as 1 and the rest as 0. The right-hand side of Fig. 5.6 shows the tensor representation of the RPCLAF algorithm where observed location–activity pairwise preferences are considered positive feedback (+1), while nonobserved location–activity pairwise preferences are marked as negative feedback (−1). All other entries are either missing (?) or zero values. For example, in the right-hand side of Fig. 5.6, the value of tensor element $\mathcal{A}_{3,1,1}$ is +1, because it holds $\mathcal{A}_{3,1,1} > \mathcal{A}_{3,1,2}$, whereas the value of tensor element $\mathcal{A}_{3,1,2} = -1$ because $\mathcal{A}_{3,1,2} < \mathcal{A}_{3,1,1}$.

5.7 Other Tensor Decomposition Methods

A drawback of TD models such as HOSVD is the fact that the construction of the core tensor requires cubic runtime in factorization dimension for both prediction and learning. Moreover, they suffer from sparsity that incurs in STSs and LBSNs. To overcome the aforementioned problem, HOSVD can be performed efficiently following the approach of Sun and Kolda [7]. Other approaches to improve the scalability to large data sets are through slicing [21] or approximation [4]. Rendle et al. [12] proposed ranking with tensor factorization (RTF), a method for learning optimal factorization of a tensor for a specific problem of tag recommendations. Moreover, Cai et al. [1] proposed LOTD, which also targets the very sparse data problem for tag recommendation. Their LOTD method is based on low-order polynomials that present low functional complexity. LOTD is capable of enhancing statistics and avoids overfitting, which is a problem of traditional tensor decompositions such as Tucker and PARAFAC decompositions. It has been experimentally shown [1] with extensive experiments on several data sets that LOTD outperforms PITF and other

methods in terms of efficiency and accuracy. Another method which outperformed PITF is proposed by Gemmell et al. [5]. Their method builds a weighted hybrid tag recommender that blends multiple recommendation components drawing separately on complementary dimensions. Moreover, Leginus et al. [11] improved tensor-based recommenders with clustering. They reduced the tag space by exploiting clustering techniques so that both the quality of recommendations and the execution time are improved.

References

1. Cai, Y., Zhang, M., Luo, D., Ding, C., Chakravarthy, S.: Low-order tensor decompositions for social tagging recommendation. In: Proceedings of the Fourth ACM International Conference on Web Search and Data Mining (WSDM'11), pp. 695–704. ACM (2011)
2. Carroll, J.D., Chang, J.J.: Analysis of individual differences in multidimensional scaling via an n-way generalization of eckart-young decomposition. Psychometrika **35**, 283–319 (1970)
3. Chen, S., Wang, F., Zhang, C.: Simultaneous heterogeneous data clustering based on higher order relationships. In: Proceedings of the Seventh IEEE International Conference on Data Mining Workshops, ICDMW'07, pp. 387–392, Washington, DC, USA. IEEE Computer Society (2007)
4. Drineas, P., Mahoney, M.W.: A randomized algorithm for a tensor-based generalization of the singular value decomposition. Linear Algebra Appl. **420**(2–3), 553–571 (2007)
5. Gemmell, J., Schimoler, T., Mobasher, B., Burke, R.: Hybrid tag recommendation for social annotation systems. In: Proceedings of the 19th ACM International Conference on Information and Knowledge Management, pp. 829–838. ACM (2010)
6. Harshman, R.A.: Foundations of the parafac procedure: models and conditions for an 'exploratory' multimodal factor analysis. In: UCLA Working Papers in Phonetics, pp. 1–84 (1970)
7. Kolda, T., Sun, J.: Scalable tensor decompositions for multi-aspect data mining. In: Proceedings of the 8th IEEE International Conference on Data Mining (ICDM'08), pp. 363–372 (2008)
8. Kolda, T.G., Bader, B.W.: Tensor decompositions and applications. SIAM Rev. **51**(3), 455–500 (2009)
9. 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
10. De Lathauwer, L., De Moor, B., Vandewalle, J.: A multilinear singular value decomposition. SIAM J. Matrix Anal. Appl. **21**(4), 1253–1278 (2000)
11. Leginus, M., Dolog, P., Žemaitis, V.: Improving tensor based recommenders with clustering. In: User Modeling, Adaptation, and Personalization Conference (UMAP 2012), pp. 151–163. Springer (2012)
12. Rendle, S., Marinho, L.B., Nanopoulos, A., Thieme, L.S.: Learning optimal ranking with tensor factorization for tag recommendation. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'09), pp. 727–736 (2009)
13. Rendle, S., Thieme, L.S.: Pairwise interaction tensor factorization for personalized tag recommendation. In: Proceedings of the Third ACM International Conference on Web Search and Data Mining (WSDM'10), pp. 81–90 (2010)
14. Rendle, S.: Context-Aware Ranking with Factorization Models, 1st edn. Springer, Berlin, Heidelberg (2010)
15. Rendle, S., Marinho, L.B., Nanopoulos, A., Schimdt-Thieme, L.: 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, pp. 727–736. ACM (2009)
16. Rendle, S., Schmidt-Thieme, L.: 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)
17. Shashua, A., Hazan, T.: Non-negative tensor factorization with applications to statistics and computer vision. In: ICML'05: Proceedings of the 22nd International Conference on Machine Learning, pp. 792–799. ACM (2005)
18. Sun, J., Shen, D., Zeng, H., Yang, Q., Lu, Y., Chen, Z.: Cubesvd: a novel approach to personalized web search. In: World Wide Web Conference, pp. 382–390 (2005)
19. Symeonidis, P., Papadimitriou, A., Manolopoulos, Y., Senkul, P., Toroslu, I.: Geo-social recommendations based on incremental tensor reduction and local path traversal. In: Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Location-Based Social Networks (LBSN), Chicago, IL, pp. 89–96 (2011)
20. Tucker, L.: Some mathematical notes on three-mode factor analysis. Psychometrika 279–311 (1966)
21. Turney, P.: Empirical evaluation of four tensor decomposition algorithms. Technical report (NRC/ERB-1152) (2007)
22. Wang, H., Ahuja, N.: A tensor approximation approach to dimensionality reduction. Int. J. Comput. Vis. 217–229 (2007)
23. Zheng, V., Cao, B., Zheng, Y., Xie, X., Yang, Q.: Collaborative filtering meets mobile recommendation: a user-centered approach. In: Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI), Atlanta, GA (2010)
24. Zheng, V., Zheng, Y., Xie, X., Yang, Q.: Towards mobile intelligence: learning from GPS history data for collaborative recommendation. Artif. Intell. **184–185**, 17–37 (2012)

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

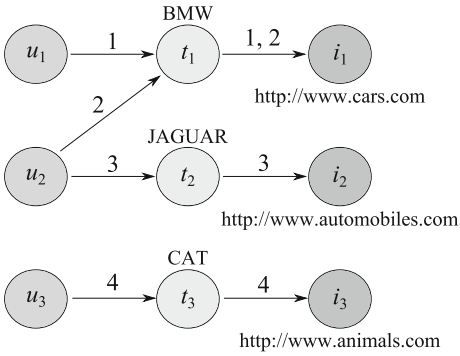

Fig. 6.1 Usage data of the running example

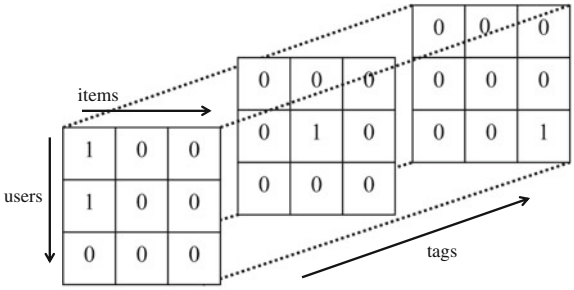

Fig. 6.2 The tensor construction of our running example

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.

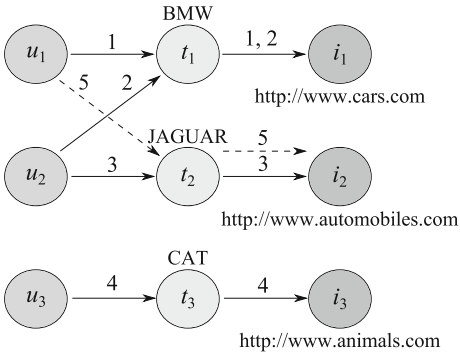

Fig. 6.3 Illustration of the tensor reduction algorithm output for the running example

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**

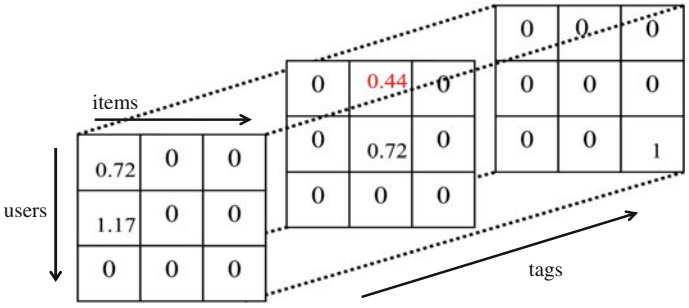

Fig. 6.4 The resulting $\hat{\mathcal{A}}$ tensor for the running example

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 ϕ. 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).

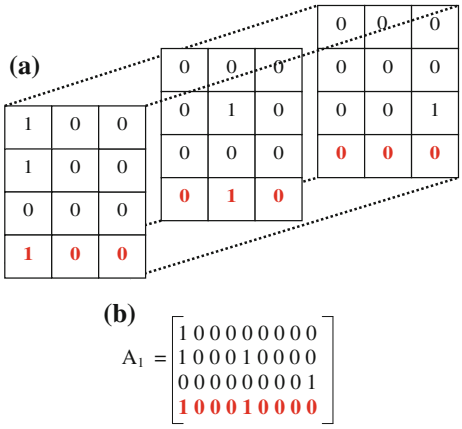

Fig. 6.5 Example of folding in a new user: **a** the insertion of a new user in tensor, **b** the new 1-mode unfolded matrix $\mathbf{A}_1$

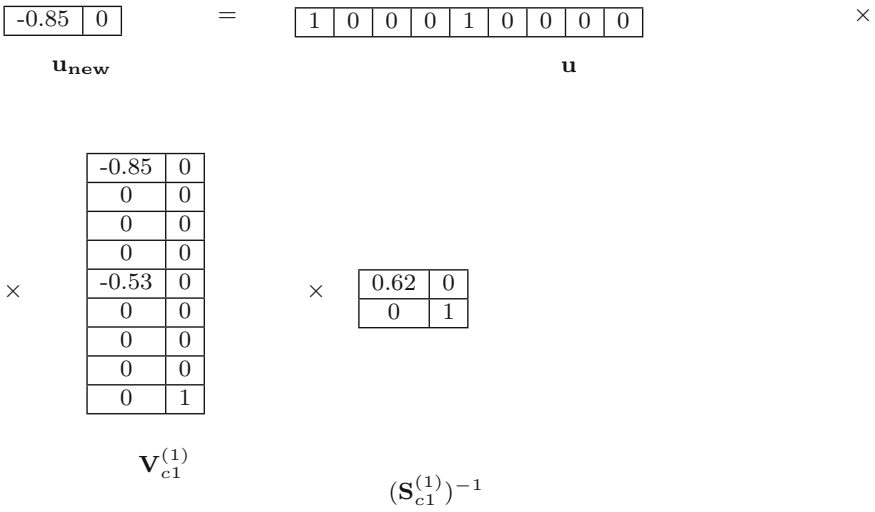

Fig. 6.6 The result of folding-in for the current example

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.

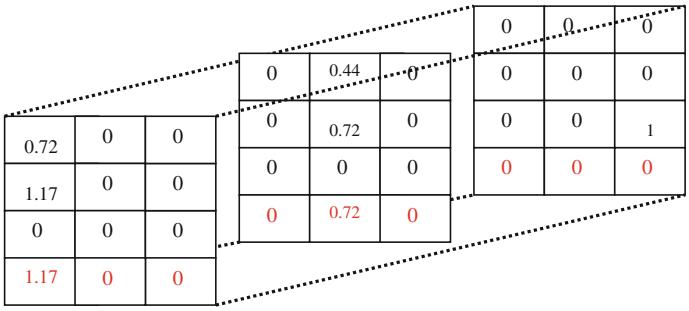

Fig. 6.7 The resulting $\hat{\mathcal{A}}$ tensor of running example after the insertion of new user

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 rela-

tionships 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)

Chapter 7
Experimental Evaluation on Tensor Decomposition Methods

Abstract In this chapter, we will provide experimental results of tensor decomposition methods on real data sets in social tagging systems (STSs). We will discuss the criteria that we will set for testing all algorithms and the experimental protocol we will follow. Moreover, we will discuss the metrics that we will use (i.e., Precision, Recall, root-mean-square error, etc.). Our goal is to present the main factors that influence the effectiveness of algorithms.

Keywords Matrix decomposition · Tensor decomposition

7.1 Data Sets

To evaluate examined algorithms, we have chosen real data sets from two different STSs: BibSonomy and Last.fm, which have been used as benchmarks in past works [3].

BibSonomy: We used a snapshot of all users, items (both publication references and bookmarks), and tags publicly available on April 30, 2007. From the snapshot, posts from the database and logic programming (DBLP) computer science bibliography are excluded since they are automatically inserted and all owned by one user and all tagged with the same tag (dblp). The number of users, items, and tags is 1,037, 28,648, and 86,563, respectively.

Last.fm: The data for Last.fm were gathered during October 2007, partly through the web services API-application program interface (collecting user nicknames), partly crawling the Last.fm site. Here, the items correspond to artist names, which are already normalized by the system. There are 12,773 triplets in the form user–artist–tag. To these triplets correspond 4,442 users, 1,620 artists, and 2,939 tags.

Following the approach of [3] to get more dense data, we adapt the notion of a p-core to tripartite hypergraphs. The p-core of level k has the property, that each user, tag, or item has/occurs in at least k posts. For both data sets we used k = 5. Thus,

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_7

for the BibSonomy data set there are 105 users, 246 items, and 591 tags, whereas for the Last.fm data set, there are 112 users, 234 items, and 567 tags.

7.2 Experimental Protocol and Evaluation Metrics

For item recommendations, all tensor decomposition algorithms have the task to predict items of users' postings in the test set. Higher Order SVD algorithm is modified appropriately to recommend items to a target user. In particular, the initial tensor represents a quadruplet $\{u, t, i, p\}$ where p is 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 a $\{u, t\}$ pair.

We performed a fourfold cross-validation, thus each time we divide the data set into a training set and a test set with sizes 75 % and 25 % of the original set, respectively. Based on the approach of [2, 4], a more realistic evaluation of recommendation should consider the division of items of each test user into two sets: (i) the *past* items of the test user and (ii) the *future* items of the test user. Therefore, for a test user, we generate recommendations based only on items in his past set. This simulates the real-world applications, where users gradually tag items and receive recommendations before they provide all their tags. As most existing works ignore this division, their reported performance corresponds to the best case, because they indirectly exploit a priori known information (items in the future set). With the division into past and future sets, accuracy is expected to decrease compared to the best case when the two sets are identical. However, reported performance is more indicative of real-world applications. The default sizes of past and future sets are 50 % and 50 %, respectively, of the number of items tagged by each test user.

As performance measures for item recommendations, we use the classic metrics of precision and recall. For a test user that receives a list of N recommended items (top-N list), precision and recall are defined as follows:

- *Precision* is the ratio of the number of relevant items in the top-N list (i.e., those in the top-N list that belong to the future set of items posted by the test user) to N.
- *Recall* is the ratio of the number of relevant items in the top-N list to the total number of relevant items (all items in the future set posted by the test user).

7.3 Sensitivity Analysis of the HOSVD Algorithm

In this section, we first conduct experiments to study the influence of core tensor dimensions on the performance of the described HOSVD algorithm. If one dimension of the core tensor is fixed, we can find that the recommendation accuracy varies as the other two dimensions change, as shown in Fig. 7.1. The vertical axes denote precision and the other two axes denote corresponding dimensions. For each figure,

one dimension is fixed and the other two dimensions are varied. Thus, for the leftmost figure, the tag dimension is fixed at 200 and the other two dimensions change. For the middle figure, the item dimension is fixed at 105. For the rightmost figure, the user dimension is fixed at 66.

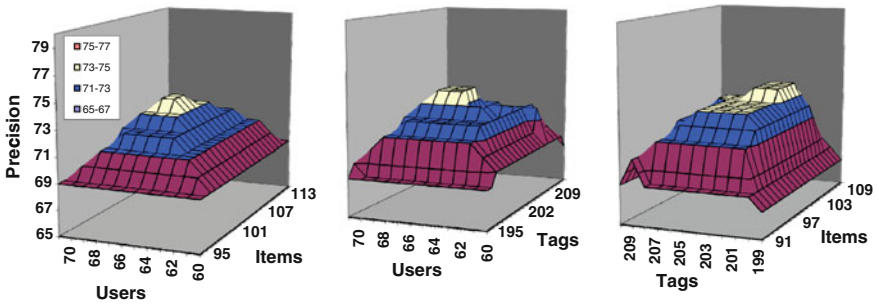

Fig. 7.1 Performance of the HOSVD algorithm as dimensions of the core tensor vary for the BibSonomy data set. For the *leftmost figure*, tag dimension is fixed at 200 and the other two dimensions change. For the *middle figure*, item dimension is fixed at 105. For the *rightmost figure*, user dimension is fixed at 66

Our experimental results indicate that a 70% of the original diagonal of $S^{(1)}$, $S^{(2)}$, $S^{(3)}$ matrices can give good approximations of A_1, A_2, A_3 matrices. Thus, the numbers c_1, c_2, and c_3 of left singular vectors of matrices $U^{(1)}$, $U^{(2)}$, $U^{(3)}$ after appropriate tuning are set to 66, 105, and 200 for the BibSonomy data set, whereas they are set to 40, 80, and 190 for the Last.fm data set.

Next, we study the influence of the proposed kernel smoothing scheme on recommendation accuracy of the HOSVD algorithm in terms of precision. We present our experimental results in Fig. 7.2a, b, for both the BibSonomy and Last.fm data sets. As shown, the smoothing kernel method can improve the performance accuracy. The results are consistent in both data sets.

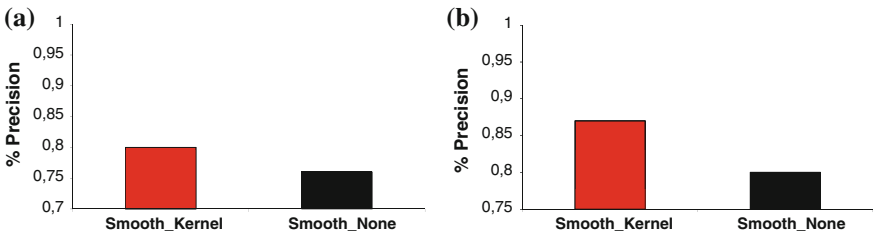

Fig. 7.2 Precision of the HOSVD algorithm associated with and without a smoothing scheme for the **a** BibSonomy data set and **b** Last.fm data set

7.4 Comparison of HOSVD with Other Tensor Decomposition Methods in STSs

In this section, we compare several tensor decomposition methods in data sets that concern the domain of STSs. To evaluate examined algorithms, we have chosen real data sets from two different social tagging systems: BibSonomy and Last.fm [7]. We have tested the following state-of-the-art methods:

- ClustHOSVD(tfidf + semantics): This is the ClustHOSVD algorithm [7], which incorporates TFIDF as a weighting schema and it is combined with the semantic similarity of tags.
- ClustHOSVD(tfidf): This is the ClustHOSVD algorithm [7], which incorporates only the term frequency-inverse document frequency.
- TFC: Rafailidis and Daras [6] proposed the Tensor Factorization and Tag Clustering model, which is a tensor factorization and tag clustering model that uses a TFIDF weighting scheme.
- HOSVD: This is the Tucker's tensor decomposition method [5].
- LOTD: Cai et al. [1] proposed low-order tensor decomposition (LOTD), which is based on low-order polynomial terms on tensors (i.e., first and second order).
- FOTD: Full Order Tensor decomposition (FOTD) proposed by Cai et al. [1] which incorporates, except the first and second terms, also the third-order polynomial term.

The parameters we used to evaluate the performance of ClustHOSVD(tfidf + semantics), ClustHOSVD(tfidf), HOSVD, Tensor Factorization and Tag Clustering model, LOTD, and FOTD are identical to those reported in the original papers. We measure precision versus recall for all six algorithms. The results for the BibSonomy and Last.fm data sets are depicted in Figs. 7.3 and 7.4, respectively.

For both data sets, ClustHOSVD(tfidf + semantics) outperforms the other comparison methods. The reason is that it exploits both the conventional cosine similarity and the semantic similarity of tags. In contrast, the Tensor Factorization and Tag Clustering model incorporates the TFIDF weighting scheme without exploiting also semantic information. FOTD presents the worst results, which are according to what

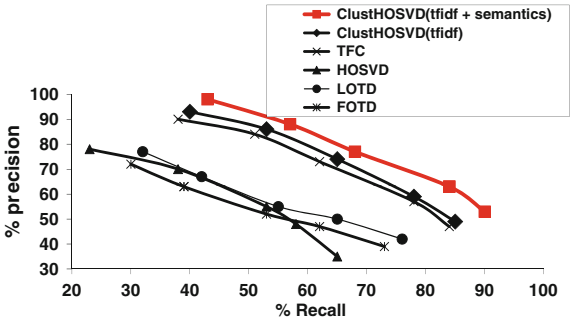

Fig. 7.3 Comparison between variations of ClustHOSVD(tfidf + semantics), ClustHOSVD(tfidf), HOSVD, and LOTD/FOTD algorithms in terms of precision–recall curve for BibSonomy data set

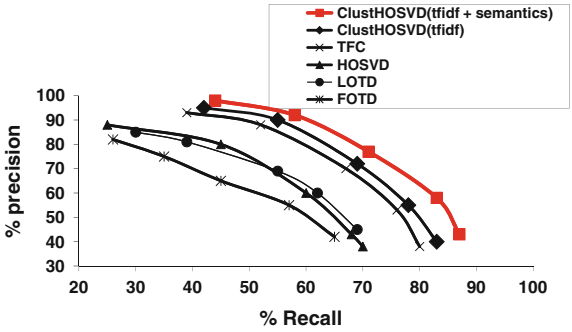

Fig. 7.4 Comparison between variations of ClustHOSVD(tfidf + semantics), ClustHOSVD(tfidf), HOSVD, and LOTD/FOTD algorithms in terms of precision–recall curve for Last.fm data set

Cai et al. [1] have reported in their paper. That is, the LOTD method had better results than FOTD in terms of precision–recall diagram, because of the overfitting problem which existed in all data sets.

References

1. Cai, Y., Zhang, M., Luo, D., Ding, C., Chakravarthy, S.: Low-order tensor decompositions for social tagging recommendation. In: Proceedings of the Fourth ACM International Conference on Web Search and Data Mining (WSDM'11), pp. 695–704. ACM (2011)
2. Herlocker, J., Konstan, J., Terveen, L., Riedl, J.: Evaluating collaborative filtering recommender systems. ACM Trans. Inf. Syst. **22**(1), 5–53 (2004)
3. Hotho, A., Jaschke, R., Schmitz, C., Stumme, G.: Information retrieval in folksonomies: search and ranking. In: The Semantic Web: Research and Applications, pp. 411–426 (2006)
4. Huang, Z., Chen, H., Zeng, D.: Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering. ACM Trans. Inf. Syst. **22**(1), 116–142 (2004)
5. De Lathauwer, L., De Moor, B., Vandewalle, J.: A multilinear singular value decomposition. SIAM J. Matrix Anal. Appl. **21**(4), 1253–1278 (2000)
6. Rafailidis, D., Daras, P.: The tfc model: tensor factorization and tag clustering for item recommendation in social tagging systems. Trans. Syst. Man Cybern.—Part A: Syst. Humans (2012)
7. Symeonidis, P.: ClustHOSVD: Item recommendation by combining semantically enhanced tag clustering with tensor HOSVD. IEEE Trans. Syst. Man Cybern. (2015)

Chapter 8
Conclusions and Future Work

Abstract In this chapter, we will discuss the main conclusions of the experimental evaluation and the limitations of each algorithm, and will provide the future research directions.

Keywords Matrix decomposition · Tensor decomposition

This book covered the major fundamentals and the key advanced topics that shape the matrix and tensor factorization field. It aimed at advanced undergraduates, graduate students, researchers, and professionals. That is, it provides researchers and developers with a comprehensive overview of the general concepts and techniques (e.g., models and algorithms) related to matrix and tensor factorization.

This book offered a rich blend of theory and practice. We have presented well-known decomposition methods for recommender systems, such as singular value decomposition, nonnegative matrix factorization, UV decomposition, Higher Order SVD, etc., to address the "information overload" problem. This problem affects our everyday experience while searching for knowledge on a topic. Naive collaborative filtering cannot deal with challenging issues such as scalability, noise, and sparsity. We have dealt with all aforementioned challenges by applying matrix and tensor decomposition methods. These methods have been proven to be the most accurate (i.e., Netflix prize) and efficient for handling big data. We described in detail the pros and cons of each method for matrices and tensors. For each method, we provided a detailed theoretical mathematical background and a step-by-step analysis, using an integrated toy example, which run throughout all chapters of the book.

We performed experiments with matrix and tensor decomposition methods in many real data sets. In particular, in matrix factorization, we compared the performance of singular value decomposition (SVD) and UV decomposition algorithms against an improved version of the original item-based collaborative filtering (CF) algorithm combined with SVD and CUR decomposition. We ran experiments on two real-life data sets (i.e., GeoSocialRec and Epinions). In tensor factorization, we provided experimental results of several tensor decomposition methods (Higher

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_8

Order SVD, ClustHOSVD, Tensor Factorization and Tag Clustering model, etc.) on two real data sets (BibSonomy, Last.fm) in STS' domain. As it is experimentally 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. However, all decomposition methods need appropriate "tuning" of the data latent feature space, to leverage the accuracy of rating prediction and recommendations.

As a future work, we could add more auxiliary information sources into our matrix and tensor decomposition models (i.e., the time dimension, the location dimension, etc.). This additional information could be used for better and more personalized recommendations, by taking advantage of the user's context-awareness.