

INSTITUT FÜR INFORMATIK  
der Ludwig-Maximilians-Universität München



Bachelorarbeit

Tag Analysis with Higher-Order SVD

Oliver Schnuck

Aufgabensteller  
und Betreuer: Prof. Dr. François Bry, Christoph Wieser  
Abgabetermin: 6. September 2010

# Erklärung

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbstständig verfasst habe und keine anderen als die angegebenen Hilfsmittel verwendet habe.

München, den 6. September 2010

Oliver Schnuck

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Latent Semantic Indexing . . . . .	4
1.2	Necessity of Higher-Order SVD . . . . .	6
<b>2</b>	<b>Mathematical Background</b>	<b>8</b>
2.1	Singular Value Decomposition . . . . .	8
2.1.1	Mathematical Introduction of SVD . . . . .	8
2.1.2	SVD in the context of LSI . . . . .	10
2.1.3	Algorithms for SVD . . . . .	11
2.2	Higher-Order Singular Value Decomposition . . . . .	12
2.2.1	Tensors - A Generalisation of Matrices . . . . .	12
2.2.2	Mathematical Introduction of HO-SVD . . . . .	15
2.2.3	Extended CubeSVD - An algorithm for HO-SVD	16
<b>3</b>	<b>Implementing a HO-SVD algorithm</b>	<b>19</b>
3.1	Programming Language . . . . .	19
3.2	Library Choice . . . . .	20
3.3	Extending the Library UJMP . . . . .	21
3.4	Implementing Extended CubeSVD . . . . .	22
<b>4</b>	<b>Validation of Tag Analysis Using HO-SVD</b>	<b>24</b>
4.1	Artigo - An Application with 3-dimensional Data . . . . .	24
4.2	Validation by Tag Recommendation . . . . .	25
4.3	Configuration Settings . . . . .	26
4.4	Results of Analysis with HO-SVD . . . . .	28
4.5	Comparison with a Two Dimensional Approach . . . . .	30
<b>5</b>	<b>Conclusion</b>	<b>33</b>
	<b>Bibliography</b>	<b>35</b>

# Abstract

a higher-Order Singular Value Decomposition is a generalisation of Singular Value Decomposition from matrices to tensors. Thus the idea is that Higher-Order Singular Value Decomposition offers the possibility to detect latent semantic structures in multidimensional data. The goal of this thesis is to validate this intuition in the context of social tagging. Therefore a sequential algorithm for computing a higher-Order Singular Value Decomposition on tensors was implemented. This implementation was used in order to analyse three-dimensional data gathered from an online game called Artigo. The quality of Higher-Order Singular Value Decomposition in terms of observing semantic structures was determined by measuring the performance of tag recommendations which resulted from this analysis. A comparison to the results of a two-dimensional approach using Singular Value Decomposition emphasises the importance of Higher-Order Singular Value Decomposition for analysing higher-dimensional data.

# Zusammenfassung

Die Singulärwertzerlegung höherer Ordnung ist eine Verallgemeinerung der Singulärwertzerlegung von Matrizen auf Tensoren. Die Idee ist nun, dass die Singulärwertzerlegung höherer Ordnung eine Möglichkeit darstellt verborgene, semantische Strukturen in mehrdimensionalen Daten zu erkennen. Ziel dieser Arbeit ist es diese Intuition im Kontext von Social Tagging zu validieren. Daher wurde ein sequentieller Algorithmus implementiert um die Singulärwertzerlegung höherer Ordnung von Tensoren zu berechnen. Die Implementierung wurde verwendet um drei-dimensionale Daten zu analysieren, die von einer Anwendung namens Artigo stammen. Die Fähigkeit der Singulärwertzerlegung höherer Ordnung, semantischen Strukturen zu erkennen, wurde untersucht, indem die Qualität von Tag Empfehlungen gemessen wurde, die aus der Analyse resultierten. Ein Vergleich mit den Ergebnissen eines zwei-dimensionalen Ansatz, der die Singulärwertzerlegung verwendet, verdeutlicht die Bedeutung der Singulärwertzerlegung höherer Ordnung für die Analyse hochdimensionaler Daten.

# Chapter 1

## Introduction

Nowadays, many applications are generating huge amounts of data, which depend on several different factors. The increasing interaction of users in the world wide web is a possible source of such multidimensional data. Detecting structures, that aren't obvious at first sight, is a task, which leads to useful results in a lot of scenarios.

A first application might be the web search market. There is a high demand for personalised web search [22] which takes into account the individual information needs of every user. Search engines accumulate huge amounts of data. This data documents, who submits queries and which pages are clicked on afterwards. This *clickthrough data* contains complicated relationships between users, queries and web pages. Detecting web users' individual interests in this recorded 3-dimensional data would be an important step to make more valuable page suggestions in future.

Social tagging is another approach which becomes more and more interesting with the rise of social networks. It describes the process, in which users add metadata in the form of keywords to arbitrary information items like songs or films. The goal of social tagging is to categorize the items and to get a common consensus about which tags fit best to an item. Tag recommendation [23] can be used to make the tagging process more comfortable for the users and thus to accumulate more data and get a more precise description of the resources. Tag analysis between these three different kinds of entities might improve the process of tag recommendation.

Before examining the analysis of multidimensional data more detailed, an introducing example is given, which motivates the need of finding

latent semantic structures. However, in this example the considered data only depends on two factors and therefore the analysis is not as complicated as in the multidimensional case.

## 1.1 Latent Semantic Indexing

In the context of information retrieval the similarity of documents to each other plays a very important role. But first of all, we have to understand how documents are formalised.

In the vector space model [21], a standard approach for information retrieval, the data structure chosen to represent a document collection is a *term-document matrix*. In a first phase, the text normalisation, each document is transformed into a set of words, which contains only those words of the document, which have an expressive meaning. Afterwards a matrix is built, that represents how often a term occurs in certain documents. The rows of this matrix correspond to the meaningful terms and the columns correspond to the documents. Consider the following documents:

Document 1: I take a picture with a Nikon camera.

Document 2: This picture isn't a painting, it's a photo made by a camera.

Document 3: Lisbon is an interesting city.

terms/docs	Doc 1	Doc 2	Doc 3
picture	1	1	0
Nikon	1	0	0
camera	1	1	0
painting	1	0	0
photo	0	1	0
Lisbon	0	0	1
interesting	0	0	1
city	0	0	1

Figure 1.1: term-document matrix containing the meaningful words

Like documents, a query can also be expressed as a vector of the terms occurring in the collection. The classical retrieval methods (i.e. similarity measures) are based on matching the terms in the user query

with the terms in the documents. Documents are considered to be similar, if and only if they have words in common. However many topics can be described in multiple ways depending e.g. on the context or people's language habits.

Let's have a closer look at this problem considering our before mentioned example. If a user query consists of the word "picture", everything is fine. As result, Document 1 and Document 2, which intuitively appear relevant for this query, are returned. However, imagine a user query consists of the word "photo", which is a synonym for "picture". Now, only Document 1 will be considered to be important and Document 2 is ignored, because it doesn't explicitly contain the word "photo".

*Latent Semantic Indexing (LSI)*, originally introduced by Deerwester et. al. [6], goes beyond full-text search and enables semantic search. It is assumed, that there is a latent semantic structure in the data, which is obscured by the randomness of word choice. LSI tries to reveal this non-obvious structure and remove the noise by identifying statistical associations of the term appearances. The statistical technique used for this goal is the *Singular Value Decomposition (SVD)*, which is explained in more detail in chapter 2.1.

SVD [14] is applied to the term-document matrix. It is possible to derive a hidden *concept* space [17] from the resulting decomposition, which associates syntactically different but semantically similar terms to concepts and contains less dimensions than the original term space. See chapter 2.1.2 for more information. The relevance of a document to a query is now measured with respect to the hidden concept space. Therefore also the query has to be transformed into the concept space in an additional step.

Let's go back to our example. Replacing the classical retrieval approach by LSI leads to a more satisfying result. The query "picture" still returns Document 1 and Document 2 as the most relevant documents. But now a query which consists of the word "photo" also returns Document 2 besides Document 1, because LSI revealed that these two documents have the same topic, which could be named "photography" in this example.

## 1.2 Necessity of Higher-Order SVD

In the previous section, the benefit of LSI in the context of information retrieval was illustrated. In other words, the statistical method Singular Value Decomposition can be used to detect latent structures in two dimensional data (e.g. terms and documents). However, if the data depends on more than two factors, Singular Value Decomposition can't be applied anymore, because it's only defined for matrices.

It is quite intuitive to store multidimensional data in multidimensional arrays, so called *tensors*. A tensor can be interpreted as a generalisation of a matrix. The following figure from [20] is the graphical representation of a tensor, which contains data depending on three different factors.

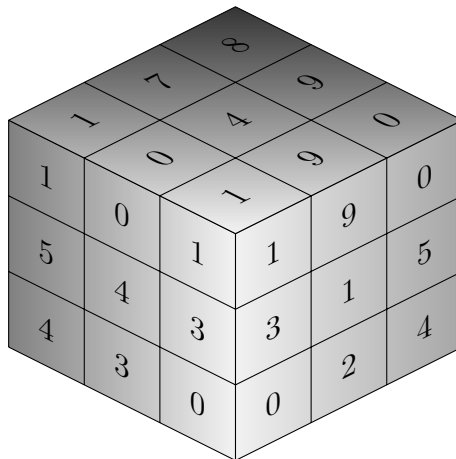


Figure 1.2: 3rd-order tensor

It is possible to generalise the concept of Singular Value Decomposition to tensors. This method is known as *Higher-Order Singular Value Decomposition (HO-SVD)* [5]. The vision of Higher-Order Singular Value Decomposition is to detect latent semantic structures in the multidimensional data, analogously to LSI respectively SVD in the 2-dimensional case.

As mentioned in [23] recent search in the area of tag analysis has focused on algorithms, which do not consider the 3 dimensions of the problem (i.e. users, items, tags) altogether. The Penalty-Reward algorithm [25] or FolkRank [15] can be mentioned in this context. They

project the three-dimensional data into three two-dimensional relations (i.e. user-item, user-tag and tag-item) and try to analyse them. But as a consequence of this projection some parts of the semantics encoded in the 3-dimensional relationships get lost. In other words some information encoded in the cube in Figure 1.2 gets lost.

Another idea would be to transform the tensor into a single matrix and analyse it with Singular Value Decomposition. However it seems likely that this compression leads to an information loss, too. Chapter 4 documents this intuition by examining a concrete data set.

Assuming that there might be even more than three factors, which determine the data, the before mentioned approaches become more and more unsuitable. In contrast to that the Higher-Order Singular Value Decomposition permits an analysis of the data without need of splitting them up into relations which ignore some parts of the multidimensional relationships. HO-SVD is defined on tensors in general. There's no restriction concerning the number of dimensions. This is another point which makes this approach interesting for a wide range of purposes.

## Chapter 2

# Mathematical Background

This chapter contains the mathematical background of the Singular Value Decomposition on matrices and the Higher-Order Singular Value Decompositions on tensors.

### 2.1 Singular Value Decomposition

#### 2.1.1 Mathematical Introduction of SVD

The definitions and explanations in this section are directly adopted from [20], which provides a more detailed elaboration of the mathematical backgrounds. The points, which are relevant for my work are presented here.

A Singular Value Decomposition (SVD) of a real matrix  $A \in \mathcal{R}^{m \times n}$  is a factorisation in three matrices  $U, \Sigma$  and  $V$  as follows [14]:

$$A = U\Sigma V^T, \Sigma = \text{diag}(\sigma_1, \dots, \sigma_p) \in \mathcal{R}^{m \times n}, p = \min\{m, n\}$$

where  $U \in \mathcal{R}^{m \times m}$  and  $V \in \mathcal{R}^{n \times n}$  are orthogonal, (i.e.  $U^T U = U U^T = I_m$ ,  $V^T V = V V^T = I_n$ ) and  $\sigma_1 \geq \dots \geq \sigma_p \geq 0$ .  $\Sigma = \text{diag}(\sigma_1, \dots, \sigma_p)$  means, that  $\Sigma$  is a (pseudo) diagonal matrix, with  $\sigma_1, \dots, \sigma_p$  in its diagonal.  $\sigma_1, \dots, \sigma_p$  are called **singular values** of  $A$ , the columns of  $U = [u_1, \dots, u_m]$  and the columns of  $V = [v_1, \dots, v_n]$  are called **left singular vectors** and **right singular vectors** of  $A$ , respectively.

Note:  $\Sigma$  is a pseudo diagonal matrix, i.e. a matrix in which nonzero elements can only occur in the diagonal of a (left) upper submatrix. Since  $\Sigma$  might have zero-valued rows and columns, which could be dropped, the SVD can also be written in this form:

$A = U\Sigma V^T$ ,  $\Sigma = \text{diag}(\sigma_1, \dots, \sigma_r) \in \mathcal{R}^{r \times r}$  with  $\sigma_r > 0$ ,  $U = [u_1, \dots, u_r] \in \mathcal{R}^{m \times r}$  and  $V = [v_1, \dots, v_r] \in \mathcal{R}^{n \times r}$ , where  $r \leq p$  is the rank of  $A$ . (Now  $\Sigma$  is a *real* diagonal matrix.) This form of a SVD is sometimes called *reduced SVD*.

**Some important facts** (from [20]):

1. Every matrix has a (reduced) Singular Value Decomposition.
2. The singular values of a matrix are uniquely determined.
3. If  $A \in \mathcal{R}^{m \times n}$  has a Singular Value Decomposition  $A = U\Sigma V^T$ , then

$$Av_j = \sigma_j u_j, \quad A^T u_j = \sigma_j v_j, \quad u_j^T Av_j = \sigma_j$$

for  $j = 1, \dots, \min\{m, n\}$ .

(These formulae show that a Singular Value Decomposition is a generalisation of an eigendecomposition. An eigendecomposition of a square matrix  $A$  is a factorisation  $A = V\Sigma V^T$ ,  $Av_j = \sigma_j v_j$ ,  $v_j^T Av_j = \sigma_j$  for  $j = 1, \dots, m$  with  $\Sigma = \text{diag}(\sigma_1, \dots, \sigma_m)$ .)

4. If  $U\Sigma V^T$  is a Singular Value Decomposition of  $A$ , then  $V\Sigma^T U^T$  is a Singular Value Decomposition of  $A^T$ .
5. If  $A \in \mathcal{R}^{m \times n}$  has  $r$  nonzero singular values, then  $\text{rank}(A) = r$  and  $A = \sum_{j=1}^r \sigma_j u_j v_j^T$
6. If  $\text{rank}(A) = r$ , then  $A$  has  $r$  nonzero singular values.
7. The nonzero singular values of  $A$  are the square roots of the nonzero eigenvalues of  $A^T A$  or  $AA^T$ .
8. If  $U\Sigma V^T$  is a (reduced) Singular Value Decomposition of  $A$ , then the columns of  $V$  are eigenvectors of  $A^T A$  and the columns of  $U$  are eigenvectors of  $AA^T$ .

An important theorem for Singular Value Decomposition is the following one:

**Eckart-Young Low Rank Approximation Theorem [14]**

Let  $A = U\Sigma V^T$  be a SVD of  $A \in \mathcal{R}^{m \times n}$  and  $r = \text{rank}(A)$ .

For  $k < r$  define  $A_k = \sum_{j=1}^k u_j \sigma_j v_j^T$ , then

$$\|A - A_k\|_F = \min_{\text{rank}(B) \leq k} \|A - B\|_F = \sqrt{\sum_{j=k+1}^r \sigma_j^2}$$

where  $\|X\|_F = \sqrt{\sum_{i,j} |x_{ij}|^2}$  is the Frobenius-norm of a matrix  $X$ .

**2.1.2 SVD in the context of LSI**

As described informally in chapter 1.1 Latent Semantic Indexing wants to derive the hidden concept space from the term-document matrix  $A \in \mathcal{R}^{m \times n}$ , which represents  $n$  documents consisting of  $m$  terms. Applying the Singular Value Decomposition to  $A$  is the mathematical technique to achieve this goal. In order to emphasise the important concepts it is necessary to eliminate the "noise" in the term-document matrix [17].

This can be done by keeping only the  $k \in [1, r]$  largest singular values of  $A$  in  $\Sigma$  and setting the others to 0. Now the size of  $\Sigma$ ,  $U$  and  $V$  can be reduced by dropping the last  $r - k$  rows and columns from  $\Sigma$ , the last  $r - k$  rows from  $V^T$  (i.e. columns from  $V$ ) and the  $r - k$  columns from  $U$  whereby we obtain  $\Sigma_k \in \mathcal{R}^{k \times k}$ ,  $U_k \in \mathcal{R}^{m \times k}$  and  $V_k \in \mathcal{R}^{n \times k}$ . The Singular Value Decomposition using  $\Sigma_k$ ,  $U_k$  and  $V_k$  leads to an approximation  $A_k$  of the original term-document Matrix  $A$ .

$$A_k = U_k \Sigma_k V_k^T$$

This is called a *truncated Singular Value Decomposition*. According to the Eckart-Young Low Rank Approximation Theorem  $A_k$  is the best rank- $k$  approximation, i.e. there's no other matrix  $M \neq A_k$  with a rank less or at most equal than  $\text{rank}(A_k) = k$  which is a better approximation of the matrix  $A$  with respect to the Frobenius-norm. This manifests the high quality of the obtained matrix  $A_k$ .

The intuition of LSI is, that SVD transforms the original  $m$ -dimensional space into a  $k$ -dimensional *concept* space [17]. The first axis of this *concept* space runs along the largest variation of the documents and corresponds to the first column of  $U_k$ , the second axis runs along the

second largest variation of the documents and corresponds to the second column, and so on. Each document was transformed by the decomposition into the new concept space and stored in  $\Sigma_k V_k^T$ . The first row in  $\Sigma_k V_k^T$  corresponds to Document 1, the second row corresponds to Document 2, and so on. The singular values in  $\Sigma_k$  are interpreted as scaling factors.

Now the similarity of documents can be measured by using their new representation in  $V_k$ . Before comparing a query with these documents, it is necessary to transform the query from the original  $m$ -dimensional space into the new *concept* space. For more details, see [17].

### 2.1.3 Algorithms for SVD

There are several different algorithms for computing a Singular Value Decomposition of matrices. In this section, two algorithms are explained briefly including their most important characteristics. The implementations of these basic algorithms are not discussed in detail. For more information, see [20].

In general there are two different kinds of approaches. On the one hand algorithms which can only be computed sequentially and therefore are called single-threaded. And on the other hand there's also a multi-threaded method, which can exploit the resources of several CPUs.

An important variant of the single-threaded algorithms is the *Basic Lanczos Method* [11]. This algorithm is based on the connection between the Singular Value Decomposition of a matrix  $A$  and the Eigenvalue Decomposition of  $A^T A$  respectively  $AA^T$  mentioned in chapter 2.1.1. Instead of directly computing the SVD of  $A$ , the Basic Lanczos Method creates a tridiagonal matrix  $T$ , which is similar to  $A^T A$ , and computes an Eigenvalue Decomposition of  $T$ . A nice feature of this algorithm is, that it doesn't require any matrix-matrix products which are quite time consuming. Hence, the approach is called matrix free. Furthermore, if an approximation of  $A$  with the  $k$  largest singular values is sufficient, this also reduces the runtime of this algorithm. Nevertheless, w.l.o.g.  $n \leq m$ , the runtime complexity of the Basic Lanczos Method is  $O(n^2 m)$  and thus has the same runtime complexity as other single-threaded approaches like e.g. the Golub-Kahan-Lanczos Method [7].

An example for a multi-threaded algorithm is *Parallel SVD* analysed

by [20]. The core of this approach is the Hestenes' Method [3], which computes an orthogonal matrix  $V$ , such that the matrix  $W = AV$  has orthogonal columns. This can be done by plane rotations. In every step several pairs of columns are orthogonalised at the same time.  $U$  is obtained by normalising the length of each nonzero column of  $W = [w_1, \dots, w_n]$  and  $\Sigma = \text{diag}(|w_1|, \dots, |w_n|)$ . Thus  $W = AV \iff A = WV^T \xrightarrow{W=U\Sigma} A = U\Sigma V^T$ . The runtime complexity depends on the number of available processors  $w$  and on a factor  $S$ . W.l.o.g.  $n \leq m$ , the runtime complexity of Parallel SVD is  $O(\lceil \frac{n}{w} \rceil Smn)$ . If  $S$  can be chosen as a constant, the runtime would be  $O(\frac{n^2 m}{w})$  which is better than the runtime complexity of the single-threaded algorithms.

A drawback of analysing two-dimensional data by Latent Semantic Indexing is the quite high runtime complexity. Therefore the Parallel SVD algorithm is an interesting alternative to the sequential approaches. However its runtime complexity depends on an unstable factor  $S$  and therefore it's not sure whether it really brings a runtime improvement without losing too much quality.

## 2.2 Higher-Order Singular Value Decomposition

### 2.2.1 Tensors - A Generalisation of Matrices

Before considering the characteristics of Higher-Order Singular Value Decomposition, it's necessary to understand the underlying idea of *tensors* and some mathematical operations connected with them. [20] gives a clear survey about the most important points and this section explicitly refers to this paper.

A tensor is a generalisation of a vector (1st order tensor) and a matrix (2nd order tensor) and thus is also called a multi-dimensional matrix. The order of a tensor  $\mathcal{A} \in \mathcal{R}^{I_1 \times \dots \times I_N}$  is  $N$ , its elements are denoted as  $a_{i_1 \dots i_n \dots i_N}$  where  $1 \leq i_n \leq I_n$  for  $1 \leq n \leq N$ .

"The *mode- $n$*  vectors of a  $N$ th order tensor  $\mathcal{A}$  are the  $I_n$ -dimensional vectors obtained from  $\mathcal{A}$  by varying the index  $i_n$  and keeping the other indices fixed. They are identical to the column vectors of the matrix unfolding  $A_{(n)} \in \mathcal{R}^{I_n \times (I_1 \dots I_{n-1} I_{n+1} \dots I_N)}$  of tensor  $\mathcal{A}$ " [22].

Let  $\mathcal{A} \in \mathcal{R}^{I_1 \times \dots \times I_N}$  be a  $N$ th-order tensor, then the *matrix unfolding*  $A_{(n)} \in \mathcal{R}^{I_n \times (I_{n+1} \dots I_N I_1 \dots I_{n-1})}$  contains the element  $a_{i_1 \dots i_N}$  of the original tensor  $\mathcal{A}$  at the position with row number  $i_n$  and column number equal

to  $(i_{n+1} - 1)I_{n+2} \dots I_N I_1 \dots I_{n-1} + (i_{n+2} - 1)I_{n+3} \dots I_N I_1 \dots I_{n-1} + \dots + (i_N - 1)I_1 \dots I_{n-1} + (i_1 - 1)I_2 \dots I_{n-1} + (i_2 - 1)I_3 \dots I_{n-1} + \dots + (i_{n-2} - 1)I_{n-1} + i_{n-1}$ . The intuition is, that the matrix unfoldings of a tensor are "matrix representations of that tensor in which all the column (row,...) vectors are stacked one after the other" [5].

In order to illustrate the somehow confusing definition for determining the column number of an element in a matrix unfolding, a simple example is given. This example and figure 2.2 are adopted from [20].

In the case of 3rd-order tensors  $\mathcal{A} \in \mathcal{R}^{I_1 \times I_2 \times I_3}$  there exist three matrix unfoldings:

- 1st-mode:  $A_{(1)} \in \mathcal{R}^{I_1 \times I_2 I_3}$ ,
- 2nd-mode:  $A_{(2)} \in \mathcal{R}^{I_2 \times I_1 I_3}$ ,
- 3rd-mode:  $A_{(3)} \in \mathcal{R}^{I_3 \times I_1 I_2}$ .

Let  $\mathcal{A} \in \mathcal{R}^{3 \times 2 \times 3}$  be a 3rd-order tensor with  $a_{111} = a_{322} = 1, a_{121} = a_{211} = a_{113} = a_{123} = 5, a_{221} = a_{311} = 9, a_{321} = a_{222} = 0, a_{122} = a_{213} = a_{313} = 7 = -a_{212}, a_{112} = 8, a_{312} = 2, a_{223} = a_{323} = 3$

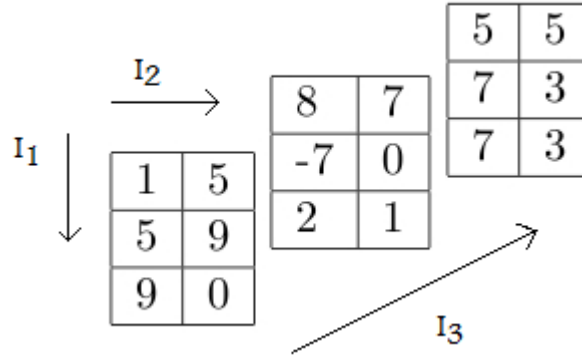
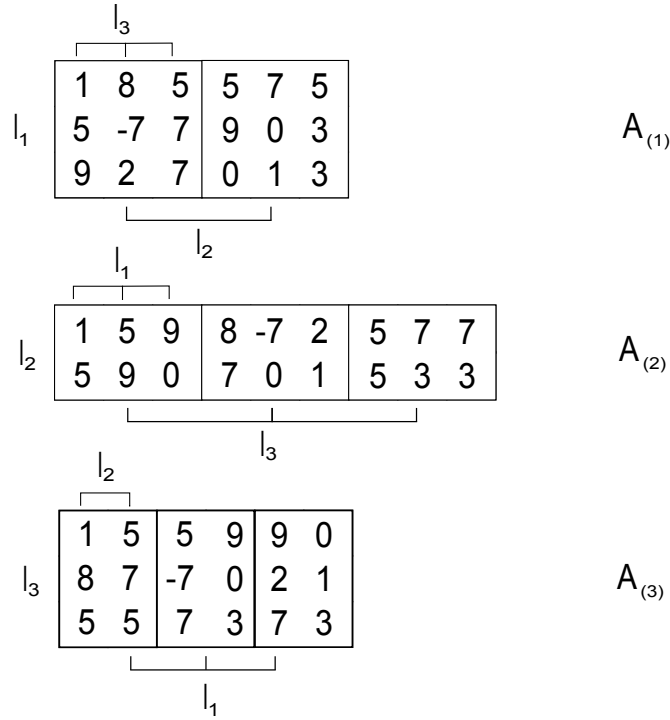


Figure 2.1: 3rd-order tensor  $\mathcal{A} \in \mathcal{R}^{3 \times 2 \times 3}$

then the matrix unfoldings  $A_{(1)} \in \mathcal{R}^{3 \times 6}, A_{(2)} \in \mathcal{R}^{2 \times 9}, A_{(3)} \in \mathcal{R}^{3 \times 6}$  have the following form:

Figure 2.2: Matrix Unfoldings of  $\mathcal{A} \in \mathcal{R}^{3 \times 2 \times 3}$ 

The *norm* of a tensor is defined by the Frobenius-norm of the unfoldings of the tensor:  $\|\mathcal{A}\| := \|A_{(1)}\|_F = \dots = \|A_{(N)}\|_F$ .

The *n-rank* of  $\mathcal{A}$ , denoted by  $R_n = \text{rank}_n(\mathcal{A})$ , is the dimension of the vector space spanned by the *n*-mode vectors and the following equation holds:  $\text{rank}_n(\mathcal{A}) = \text{rank}(A_{(n)})$ .

The *n-mode product*  $\times_n$  of a tensor and a matrix is defined as follows: Let  $\mathcal{A} \in \mathcal{R}^{I_1 \times \dots \times I_N}$  and  $M \in \mathcal{R}^{J_n \times I_n}$  then  $\mathcal{A} \times_n M := \mathcal{N}$  with  $\mathcal{N} \in \mathcal{R}^{I_1 \times \dots \times I_{n-1} \times J_n \times I_{n+1} \times \dots \times I_N}$  and  $(\mathcal{N})_{i_1 \dots i_{n-1} j_n i_{n+1} \dots i_N} = \sum_{i_n \in I_n} a_{i_1 \dots i_N} m_{j_n i_n}$

The following calculation rules are valid for the n-mode product:

- $\mathcal{A} \in \mathcal{R}^{I_1 \times \dots \times I_N}$ ,  $M_1 \in \mathcal{R}^{J_n \times I_n}$ ,  $M_2 \in \mathcal{R}^{J_m \times I_m}$ ,  $m \neq n$   
 $\Rightarrow (\mathcal{A} \times_n M_1) \times_m M_2 = (\mathcal{A} \times_m M_2) \times_n M_1 = \mathcal{A} \times_n M_1 \times_m M_2$ , [5]  
 (This equation could be seen as a kind of associativity of the *n*-mode product.)
- $\mathcal{A} \in \mathcal{R}^{I_1 \times \dots \times I_N}$ ,  $M_1 \in \mathcal{R}^{J_n \times I_n}$ ,  $M_2 \in \mathcal{R}^{K_n \times J_n}$ ,  
 $\Rightarrow (\mathcal{A} \times_n M_1) \times_n M_2 = \mathcal{A} \times_n (M_2 M_1)$ , [5]

(This equation shows the relation between the  $n$ -mode product and the matrix-matrix product.)

The SVD of a matrix  $A \in \mathcal{R}^{I_1 \times I_2}$  can be rewritten in terms of  $n$ -mode products:  $A = U\Sigma V^T = \Sigma \times_1 U \times_2 V$  with the properties of the SVD described in the previous section.

There are some other mathematical operations in the context of tensors like the scalar product or the tensor product. Since they are not relevant for the Higher-Order SVD, they aren't discussed here.

### 2.2.2 Mathematical Introduction of HO-SVD

*Higher-Order Singular Value Decomposition (HO-SVD)* is a generalisation of the Singular Value Decomposition and can be applied to tensors. It was proposed by De Lathauwer et al. [5]. The Higher-Order Singular Value Decomposition (or the  $N$ th-order SVD) of a tensor  $\mathcal{A} \in \mathcal{R}^{I_1 \times \dots \times I_N}$  is:

$$\mathcal{A} = \mathcal{S} \times_1 U_1 \times_2 \dots \times_N U_N$$

with

- $U_n = [u_{n,1}, \dots, u_{n,n}] \in \mathcal{R}^{I_n \times I_n}$  (with  $u_{n,i} \in \mathcal{R}^{I_n \times 1}$ ) is orthogonal for  $1 \leq n \leq N$ .
- $\mathcal{S} \in \mathcal{R}^{I_1 \times \dots \times I_N}$ , where the subtensors  $\mathcal{S}_{i_n=a}$  (i.e.  $\mathcal{S}$  with fixed  $n$ th index to  $a$ ) are *all-orthogonal* (i.e. two different subtensors  $\mathcal{S}_{i_n=a}$  and  $\mathcal{S}_{i_n=b}$  are orthogonal for all possible values of  $n$ :  $\langle \mathcal{S}_{i_n=a}, \mathcal{S}_{i_n=b} \rangle = 0$  for  $a \neq b$ ) and *ordered* (i.e.  $\|\mathcal{S}_{i_n=1}\| \geq \dots \geq \|\mathcal{S}_{i_n=I_n}\| \geq 0$  for all possible values of  $n$ ).

$\sigma_{n,i} := \|\mathcal{S}_{i_n=i}\|$  are called  *$n$ -mode singular values* and the vectors  $u_{n,i} \in \mathcal{R}^{I_n}$  are called  *$n$ -mode singular vector* of  $\mathcal{A}$ .  $\mathcal{S}$  is called the *core tensor*. Note some characteristics of Higher-Order SVD [20]:

- Every tensor has a higher-Order Singular Value Decomposition.
- The  $n$ -mode singular values are uniquely defined.
- The  $n$ -mode singular vectors of a tensor are the generalisation of the left and right singular vectors of a matrix.
- The value of the Frobenius-norm of the  $(N-1)$ th-order subtensors of the core tensor  $\mathcal{S}$  is taking over the role of the singular values.

- In the matrix case, the singular values also correspond to the Frobenius-norm of the rows and the columns of the 'core matrix'  $\mathcal{S}$  and  $\mathcal{S}$  is all-orthogonal, because it is diagonal.

An important link between HO-SVD and SVD is the following **theorem** [20]:

Let  $\mathcal{A} = \mathcal{S} \times_1 U_1 \times_2 \dots \times_N U_N$  a HO-SVD, then the SVD of the n-mode matrix unfolding  $A_{(n)}$  is:  $A_{(n)} = U_n \Sigma_n V_n^T$ , with  $\Sigma_n = \text{diag}(\sigma_{n,1}, \dots, \sigma_{n,I_n}) \in \mathcal{R}^{I_n \times I_n}$  and  $V_n^T := (\Sigma_n^{-1} S_n) \cdot (U_{n+1} \otimes \dots \otimes U_N \otimes U_1 \otimes \dots \otimes U_{n-1})^T \in \mathcal{R}^{I_{n+1} \dots I_N I_1 \dots I_{n-1} \times I_n}$ , where  $\otimes$  denotes the Kronecker product.

Without studying the details of the Kronecker product, this means it is possible to compute a HO-SVD of a n-mode tensor  $\mathcal{A}$  by computing SVDs of its matrix unfoldings  $A_{(1)}, A_{(2)}, \dots$  and  $A_{(n)}$  and using the resultant matrices containing the left singular vectors.

### 2.2.3 Extended CubeSVD - An algorithm for HO-SVD

As mentioned in the last section, De Lathauwer et al. [5] sketched a way, how to compute the Higher-Order SVD of a tensor by computing the SVD of its unfoldings. *CubeSVD* [22] summarises this approach of De Lathauwer. However, it is only possible to compute the Higher-Order SVD of 3rd-order tensors with CubeSVD. Therefore a novel formalisation of the approach of De Lathauwer for computing the Higher-Order SVD of  $n$ th-order tensors is presented now. We call it *Extended CubeSVD*. Since the implementation, which is described in the following chapter, refers to this algorithm, it is described more detailed.

Input for the algorithm is a  $n$ th-order Tensor  $\mathcal{A} \in \mathcal{R}^{d_1 \times d_2 \times \dots \times d_n}$ . CubeSVD admits the user to retain the best rank- $(a_1, a_2, \dots, a_n)$  approximation  $\mathcal{A}_{(a_1, a_2, \dots, a_n)}$  for  $\mathcal{A}$ . Similar to Latent Semantic Indexing, the background of such an approximation is the goal to remove the "noise" in the data. The design of the following listing is inspired from [20].

**Algorithm: Extended CubeSVD****Input:**  $n$ th-order tensor  $\mathcal{A} \in \mathcal{R}^{d_1 \times d_2 \times \dots \times d_n}$ with  $(a_1, a_2, \dots, a_n) \in [1, d_1] \times [1, d_2] \times \dots \times [1, d_n]$ **Output:**  $n$ th-order tensor  $\mathcal{A}_{(a_1, a_2, \dots, a_n)} \in \mathcal{R}^{d_1 \times d_2 \times \dots \times d_n}$   
(an approximation to  $\mathcal{A}$ )

---

```

1  for  $i = 1, \dots, n$ 
2    Calculate the matrix unfolding  $A_{(i)}$ 
3  end for
4  for  $i = 1, \dots, n$ 
5    Compute SVD of  $A_{(i)}$  with  $A_{(i)} = U_i \Sigma_i V_i^T$ 
6  end for
7  for  $i = 1, \dots, n$ 
8    Let  $W_i = [u_{i,1}, \dots, u_{i,a_i}]$  with  $u_{i,j}$  column vectors of  $U_i$ .
9  end for
10 Compute  $\mathcal{S} = \mathcal{A} \times_1 W_1^T \times_2 W_2^T \times_3 \dots \times_n W_n^T$ 
11 Compute  $\mathcal{A}_{(a_1, a_2, \dots, a_n)} = \mathcal{S} \times_1 W_1 \times_2 W_2 \times_3 \dots \times_n W_n$ 

```

---

Here are some interesting explanations and notes for this algorithm:

- $A_{(1)} \in \mathcal{R}^{d_1 \times d_2 d_3 \dots d_n}$ ,  $A_{(2)} \in \mathcal{R}^{d_2 \times d_1 d_3 d_4 \dots d_n}$ , ...,  $A_{(n)} \in \mathcal{R}^{d_n \times d_1 d_2 \dots d_{n-1}}$
- $U_1 \in \mathcal{R}^{d_1 \times d_1}$ ,  $U_2 \in \mathcal{R}^{d_2 \times d_2}$ , ...,  $U_n \in \mathcal{R}^{d_n \times d_n}$
- $W_1 \in \mathcal{R}^{d_1 \times a_1}$ ,  $W_2 \in \mathcal{R}^{d_2 \times a_2}$ , ...,  $W_n \in \mathcal{R}^{d_n \times a_n}$
- $\mathcal{A} \times_1 W_1^T \in \mathcal{R}^{a_1 \times d_2 \times \dots \times d_n}$ ,  $\mathcal{A} \times_1 W_1^T \times_2 W_2^T \in \mathcal{R}^{a_1 \times a_2 \times \dots \times d_n}$ , ...,  
 $\mathcal{S} = \mathcal{A} \times_1 W_1^T \times_2 W_2^T \times_3 \dots \times_n W_n^T \in \mathcal{R}^{a_1 \times a_2 \times \dots \times a_n}$
- $\mathcal{S} \times_1 W_1 \in \mathcal{R}^{d_1 \times a_2 \times \dots \times a_n}$ ,  $\mathcal{S} \times_1 W_1 \times_2 W_2 \in \mathcal{R}^{d_1 \times d_2 \times a_3 \times \dots \times a_n}$ , ...,  
 $\mathcal{A}_{(a_1, a_2, \dots, a_n)} = \mathcal{S} \times_1 W_1 \times_2 W_2 \times_3 \dots \times_n W_n \in \mathcal{R}^{d_1 \times d_2 \times \dots \times d_n}$
- $a_1, a_2, \dots, a_n$  are reducing the dimensionality of  $U_1, U_2, \dots, U_n$ , respectively. These numbers "are chosen by preserving a percentage of information of the original  $\Sigma_1 = S_1, \Sigma_2 = S_2, \dots, \Sigma_n = S_n$  matrices after appropriate tuning (the default percentage is set to 50% of the original matrix)." [23]
- Any (non-randomised) algorithm to compute the matrix SVD could be used in line 5.
- If  $a_1 = d_1, a_2 = d_2, \dots, a_n = d_n$  then the approximation tensor  $\mathcal{A}_{(a_1, a_2, \dots, a_n)} = \mathcal{A}_{(d_1, d_2, \dots, d_n)}$  of  $\mathcal{A}$  is exactly  $\mathcal{A}$ .

Now the runtime complexity of Extended CubeSVD is considered.

W.l.o.g.  $d_1 \geq d_2 \geq \dots \geq d_n$

Computing the unfolding in line 2 is a matter of reordering the indices in the correct way [20]. For each element in the tensor the corresponding column value in the unfolding has to be computed. Without a more detailed examination, this calculation is supposed to need  $O(n^2)$ . Thus line 2 has a runtime complexity of  $O(n^2 d_1 d_2 \dots d_n)$  and lines 1-3 need  $n \cdot O(n^2 d_1 d_2 \dots d_n) = O(n^3 d_1 d_2 \dots d_n)$ . The computational cost of lines 4-6 is  $O(d_1^2 d_2 d_3 \dots d_n + d_2^2 d_1 d_3 d_4 \dots d_n + \dots + d_n^2 d_1 d_2 \dots d_{n-1}) = n \cdot O(d_1^2 d_2 d_3 \dots d_n) = O(n d_1^2 d_2 d_3 \dots d_n)$  due to the runtime complexity of SVD on matrices. Line 8 can be computed in constant time, therefore lines 7-9 needs  $O(n)$ . The n-mode product of a tensor  $\mathcal{T} \in \mathcal{R}^{I_1 \times I_2 \times \dots \times I_n}$  and a matrix  $\mathcal{M} \in \mathcal{R}^{G_j \times I_j}$  has a runtime complexity of  $O(I_1 I_2 \dots I_3 G_j)$ . The reason for that is that the resulting matrix has, e.g. for  $j = 2$ ,  $I_1 G_j I_3 I_4 \dots I_n$  entries and for each entry  $I_2$  products have to be summed up. Thus line 10 and 11 both have a runtime complexity of  $O(d_1 d_2 \dots d_n a_1 + a_1 d_2 d_3 \dots d_n a_2 + \dots + a_1 a_2 \dots a_{n-1} d_n a_n) = O(d_1^2 d_2 d_3 \dots d_n + d_2^2 d_1 d_3 d_4 \dots d_n + \dots + d_n^2 d_1 d_2 \dots d_{n-1}) = n \cdot O(d_1^2 d_2 d_3 \dots d_n) = O(n d_1^2 d_2 d_3 \dots d_n)$ . Finally the runtime complexity of the whole algorithm is  $O(n d_1^2 d_2 d_3 \dots d_n + n^3 d_1 d_2 \dots d_n)$ . In most of the applications of HO-SVD  $n$  is a constant or at least significantly smaller than  $\sqrt{d_1}$ . Under this condition the runtime can be stated as  $O(n d_1^2 d_2 d_3 \dots d_n)$  and is mainly determined by the complexity both of Singular Value Decomposition and the n-mode product. For  $n = 3$  the time complexity is  $O(d_1^2 d_2 d_3)$ .

Besides [20] proposes a multi-threaded algorithm called *Parallel Higher-Order Singular Value Decomposition*. It is based on CubeSVD and therefore only applicable to 3rd-order tensors. However it is possible to extended it to higher-order tensors similar to the algorithm above. In line 5 the Parallel SVD algorithm is used for the Singular Value Decomposition of the unfoldings. Furthermore the n-mode Products in Line 10 and 11 are parallelised. This leads to a runtime complexity of  $O(\frac{d_1^2 d_2 d_3 + d_2^2 d_1 d_3 + d_3^2 d_1 d_2}{w} + \lceil \frac{d_1}{w} \rceil S d_1 d_2 d_3) = O(S \frac{d_1^2 d_2 d_3}{w})$  with  $w$  being the number of available processors.  $S$  is a factor, which is according to [3] either a constant or  $O(\log(n))$ . Hence it isn't sure whether Parallel Higher-Order SVD is a better algorithm in terms of runtime complexity than CubeSVD, because it depends on the number of available processors and the property of  $S$ .

## Chapter 3

# Implementing a HO-SVD algorithm

Having discussed the theoretical and mathematical aspects of Higher-Order Singular Value Decomposition, it's now possible to take a closer look at implementing a concrete algorithm for this purpose. Extended CubeSVD, explained in the previous section, is the basis for the implementation.

### 3.1 Programming Language

A precondition for implementing an algorithm is choosing an appropriate programming language. From the beginning, a goal behind implementing a higher-Order SVD was to analyse multidimensional data gathered from a game called Artigo [16]. For more information about Artigo, see chapter 4.1. There already existed an application in the context of Artigo written in *Java*, which should be able to access the algorithm and gain the results of it. Hence a programming language compatible to *Java* was preferred.

Besides implementing a single-threaded algorithm for computing Higher-Order SVD, it seemed possible that the sequential approach might once be extended to a multi-threaded algorithm. Thus in a first attempt *Scala* [19] was used, a quite new programming language which contains a sophisticated approach for parallel programming, the *actor model*. Furthermore it combines characteristics of the object-oriented and the functional programming paradigm, which is an advantage for implementing the algorithm. However, the announced interoperability with *Java* turned out to be deficient in some points. Especially the interaction with the external *Java* library UJMP revealed several

difficulties. Therefore the decision was made in favour of *Java*.

## 3.2 Library Choice

Since Higher-Order SVD is arranged in the context of linear algebra and *Java* doesn't provide appropriate data structures in the JDK, an external library had to be chosen. Furthermore the library should be able to compute a Singular Value Decomposition on matrices, because this is an essential part of computing a HO-SVD.

There are several libraries, which were developed for this purpose, with *Jama* [18] and *Colt* [4] being the most common. Recently, another promising open source source library has been released, called *Universal Java Matrix Library (UJMP)* [1].

All of them implement the same SVD algorithm, which was adopted from the *Fortran* library *Linpack* [8]. It's a variant of the algorithm published by Golub and Reinsch [10] and computes the full set of singular values simultaneously. However, this is the reason why a truncated SVD, i.e. using only the  $k$  largest singular values, attains no runtime improvement in comparison to the full decomposition [26]. In Opposite to that a SVD implementation using the Basic Lanczos Method ensures a runtime improvement in such a case. Therefore replacing the used SVD algorithm by a Basic Lanczos Method might improve the practical runtime performance of our Higher-Order SVD implementation. PROPACK written in Fortran and Matlab or the Fortran library SVDPACK, which also exists as a Java translation, may be an appropriate choice. For more details about the SVD algorithm used in our implementation, have a look at [9].

*Jama* provides a standard implementation for two-dimensional double matrices and its mathematical decompositions. However the matrices are always internally represented as two-dimensional arrays [12]. A sparse matrix is usually described as a matrix where "many" of its elements are equal to zero. Matrices in the context of social tagging are usually sparse. Representing huge, sparse matrices in such a data structure leads to problems concerning the space. Furthermore there's no possibility to represent higher-order tensors. In opposite to that, *Colt* permits the storage of sparse matrices. Furthermore it offers possibility to represent 3rd-order tensors.

Nevertheless *UJMP* [2] outperforms both *Colt* and *Jama*. First of all

it also supports tensors with an order higher than 3. This enables an extension of CubeSVD to higher-order tensors. *UJMP* has a flexible architecture, which can easily be extended and accessed by other libraries, e.g. *Colt*. Besides, it provides interfaces to external data sources such as SQL databases. This is an useful feature because the data, which is analysed in chapter 4, is stored in a Postgres database. Although the documentation of the library is hardly sufficient, the named features are useful, strong advantages. Thus *UJMP* was chosen to be the appropriate matrix library. The implementation of HO-SVD refers to Version 0.2.4 of *UJMP*.

### 3.3 Extending the Library UJMP

Basic matrix operations like the matrix product or transposing a matrix are already implemented in *UJMP*. However, although it provides a data structure for higher-dimensional tensors, which is internally realised by a hashmap, basic tensor operations like the n-mode product or the unfolding of a tensor are still missing. Therefore it was necessary to implement them manually.

The correctness of the implementation of the n-mode product was validated experimentally by showing that the associativity of the n-mode product, mentioned in chapter 2.2.1, holds for different examples. The only way to check the correctness of the unfolding was to examine some examples taken from other articles. Furthermore, recomputing a tensor which was decomposed by the Higher-Order SVD implementation with full rank, results in a tensor with almost the same values (neglecting insignificant differences which occur because of the SVD algorithm being a numerical procedure). This also indicates, that the basic operations are implemented correctly.

The Java class of *UJMP*, which enables access to higher-order tensors stored in a Postgres database is called `JDBCSParseObjectMatrix`. This class doesn't transfer the tensor from the database to the main memory. It's simply an interface to access the values in the database with an SQL statement each time they are required. However, this leads to a rather slow runtime behaviour of the HO-SVD algorithm. Therefore a method called `toSparseMatrix()` was implemented, which converts the tensor specified in the database to the main memory. Furthermore, an erroneous behaviour of the class `JDBCSParseObjectMatrix` was detected. The method `availableCoordinates()` of the class `JDBCSParseObjectMatrix`, which should only return the coordinates

of entries, which are explicitly specified in the database, actually returns the coordinates of all entries. This was corrected in the class `MyJDBCsparseObjectMatrix`, which extends `JDBCsparseObjectMatrix` and also contains the method `toSparseMatrix()`.

### 3.4 Implementing Extended CubeSVD

As already mentioned, the implemented Higher-Order SVD algorithm mainly sticks to Extended CubeSVD explained in chapter 2.2.3.

The algorithm is specified in the class `ExtendedCubeSVD`. The design of this class is analogue to the already existing class `SVD.SVDMatrix`, which computes the SVD of a two-dimensional matrix. The decomposition is calculated by constructing an object of this class. The first argument of the constructor is the tensor  $\mathcal{A} \in \mathcal{R}^{d_1 \times d_2 \times \dots \times d_n}$  on which HO-SVD should be computed. Then a variadic list of integer values follows. These integers specify the approximation level of the resulting tensor. The first integer corresponds to  $a_1 \in [1, d_1]$  in the listing of Extended CubeSVD in chapter 2.2.3, the second integer corresponds to  $a_2 \in [1, d_2]$  and so on. For tensors with higher order than 2 the number of integers has to be equal to the order of the tensor in the first argument.

The constructor of the class computes the core tensor  $\mathcal{S}$  and the  $n$ -mode singular vectors in form of  $W_i$  for each  $i \in [1, n]$ , i.e. it basically executes lines 1 to 10 of the listing. These tensors are specified as fields of the class and can be accessed after the object is constructed by using the corresponding methods `getS()`, `getW()` or `getWArray()`.

In order to retain the approximated tensor  $\mathcal{A}_{(a_0, a_1, \dots, a_n)}$  the method `computeApprox` has to be called. Its execution corresponds to line 11 of the listing.

The class `SVD.SVDMatrix` of the UJMP library computes a SVD on two-dimensional matrices. However, it doesn't provide a possibility to compute a truncated SVD on a matrix, using only the  $k$  largest singular values. Therefore a possibility to calculate a truncated SVD is integrated in the class `ExtendedCubeSVD`, too. This can be done by calling the constructor with a two-dimensional matrix  $\mathcal{A} \in \mathcal{R}^{m \times n}$  as first argument, and an integer  $k \in [1, \min(m, n)]$  as second argument. In conformity with chapter 1.1,  $\mathcal{U}_k$  can afterwards be accessed by the method `getU`,  $\mathcal{V}_k$  can afterwards be accessed by the method `getV` and

$\Sigma$  can be accessed by *getS*. The approximated matrix  $\mathcal{A}_k$  is calculated by the method *computeApprox()*.

## Chapter 4

# Validation of Tag Analysis Using HO-SVD

The implementation of the Higher-Order SVD algorithm described in the last chapter is now used in order to analyse three-dimensional data gathered by an application called Artigo<sup>1</sup>. Goal of this process is to examine whether the Higher-Order SVD is able to capture the most important structures in the data ignoring the noise. This evaluation takes place in the context of tag recommendation.

### 4.1 Artigo - An Application with 3-dimensional Data

*Artigo* is a project stated in the area of art history. The basic idea of Artigo [16] is to make pictures more meaningful by assigning them tags, which refer to their content, form, colours and other characteristics. However, yielding this task to single individuals is likely to result in inadequate tags. Thus Artigo always involves two users at the same time in the tagging process.

Artigo is based on the *ESP game*, which was introduced by Luis von Ahn [24] and later adopted by *Google Image Labeler*. Two randomly selected online-users take part in one round during which they get to see the same sequence of pictures. If a user doesn't want to continue with a picture, he can skip to the next one. From the player's perspective, goal of the Artigo game is to guess what his partner is typing for each image. If you assign the same tag to a picture like your partner, both of you are rewarded by one point.

---

<sup>1</sup><http://www.artigo.org>

In order to get a various set of tags for each image, an additional constraint is made. The players are confronted with taboo words that have previously been chosen and for which you can not win points anymore. You will get a higher score for finding a new expression for pictures that have such taboo words. The more taboo words a picture has the more points the users get for matching terms.

In this way 3-dimensional data is produced, which is stored in a 3-dimensional tensor  $\mathcal{O} \in \mathcal{R}^{d_1 \times d_2 \times d_3}$ , whereby  $d_1$  corresponds to the number of users which took part in a game,  $d_2$  corresponds to the number of pictures which were tagged and  $d_3$  is the number of used tags. Each entry of the resulting tensor  $\mathcal{A}$  is either 0 or 1. Entry  $a_{xyz}$  has the value 1, if and only if user  $x$  assigned tag  $z$  to picture  $y$  and the tagging attended to the rules of the game.

## 4.2 Validation by Tag Recommendation

Obviously Artigo is located in the area of social tagging. Users add metadata in form of keywords to pictures. The resulting 3-dimensional data is analysed using the implementation of the Extended CubeSVD algorithm from chapter 3.4. In order to measure the quality of this analysis we are using tag recommendation [23].

The original data set, which is represented by the entries in the 3-dimensional tensor  $\mathcal{T}$  with the value 1, is divided into two distinct sets: a training set and a test set. The algorithm has the task to predict the users' postings in the test set only by analysing the training set. This is done by computing an approximation of the tensor, which corresponds to the training set using Extended CubeSVD. The setting that the test set isn't part of the training set is necessary, because in real-world applications users gradually tag items and there's no access to all the taggings of the users including the future ones. However, according to [23] some existing works don't stick to this division and therefore might gain better but unrealistic results.

Now the question is how to interpret the resulting HO-SVD of the training tensor  $\mathcal{T} \in \mathcal{R}^{d_1 \times d_2 \times d_3}$  in order to decide, which tags should be recommended to a user for a certain picture. Applying Extended CubeSVD to  $\mathcal{T}$  leads to a 3-dimensional tensor  $\mathcal{T}'_{(a_1, a_2, a_3)}$  that approximates  $\mathcal{T}$ . [23] postulates that  $t'_{xyz}$  can be interpreted as the likeliness that user  $x$  tags item  $y$  with the keyword  $z$ . Therefore to recommend

the most relevant  $N$  tags for user  $x$  and picture  $y$ , the tags with the  $N$  highest values  $t'_{xyz}$  for  $z \in [1, d_3]$  are chosen.

At last we have to consider how to measure the quality of a list of  $N$  recommended tags (top- $N$  list) for a fixed user and a fixed picture. Of course tags, which were already assigned from user  $x$  to picture  $y$  in the training, shouldn't be part of the top- $N$  list. For this purpose we are using the popular metrics *precision* and *recall* [13].

Therefore the tags must be separated with respect to two different aspects. On the one hand it has to be decided whether a tag is relevant or not. In our context, a tag  $z$  is relevant for a user  $x$  and a picture  $y$ , if it is part of the test set [23]. On the other hand, the tags have to be divided into the set that was recommended respectively selected to a user for an item and the set that was not. Let  $N_r$  be the number of relevant tags and  $N_{rs}$  the number of relevant and selected tags.

Then the precision  $P$  is defined as the ratio of the number of relevant tags in the top- $N$  list to the number of recommended tags.

$$P = \frac{N_{rs}}{N}$$

The Recall  $R$  of a recommendation is the ratio of the number of relevant tags in the top- $N$  list to the total number of relevant tags.

$$R = \frac{N_{rs}}{N_r}$$

This is how the quality of a single top- $N$  list for a given user and a given resource is measured. In order to estimate the quality of the recommendations of the whole test set the same metrics are used. The precision respectively the recall of each element of the test set is computed and then the arithmetic mean of these values is built.

### 4.3 Configuration Settings

The data set retained from Artigo consists of 1739054 taggings, i.e. 3-ary tuples, which consist of a user, a picture and a tag. They are stored in a Postgres database. These taggings were made by 1794 persons about 23013 pictures using 78016 different tags. However, our implementation of Extended CubeSVD faces some problems in dealing with such a huge amount of data. The runtime of the algorithm would be far too high and besides further questions concerning the storage of

the data would have to be answered. It might be a future challenge to concentrate on these aspects. But the goal of this work is to examine the efficiency of analysing multidimensional data with Higher-Order SVD in general.

Therefore (as previous works do [23], [22]) we focus on a smaller set as the one stored in the database. We determined the 100 users which entered the most tags, the 100 pictures which received at most tags and the 100 tags which were used most often in order to describe a picture. Then all the taggings, which were made by one of these users for one of these pictures using one of these tags, were filtered. This selection was made in order to get dense data. The resulting tensor  $\mathcal{O} \in \mathcal{R}^{100 \times 100 \times 100}$  contains 6817 entries with the value 1 and is called original tensor.

As already mentioned in the previous section, the original set, which constitutes the original tensor  $\mathcal{O}$ , has to be divided into two distinct, complementary sets. The size of the training set is determined to be 90% of the original set and thus the size of the test set is 10% of the original set. In addition to the ratio of the size of these two sets also the selection, which taggings belong to the trainings set and which ones to the test set, influences the analysis results.

A first variant is to randomly select 10% of the taggings in the original set and declare them as the test set. Since the original set is stored in a Postgres database, the division was made using the *random()* function of Postgres. The remaining 90% of the original set build the training set. We call this selection *random selection*.

Another possibility to partition the original set is to order the original set with respect to the creation time of the taggings. This information is available for the Artigo data set. Then the newest 10% of the taggings are assigned to the test set and the other 90% build the training set. This division is quite reasonable, because it addresses real-world applications, in which tag recommendations can only be made based on former taggings. We call this selection *temporal selection*.

Of course there are several other ways to divide the original set into a training set and a test set. However the following validations just refer to the two mentioned ones, because they seem to be intuitive and reasonable.

#### 4.4 Results of Analysis with HO-SVD

As already mentioned we determine the precision respectively the recall of the top- $N$  list of each element in the test set and build the arithmetic mean of these values. In order to characterise the quality of the tag analysis more detailed, precision and recall are calculated for  $N \in [1, 5]$ .

a higher-Order SVD is computed for the training tensor  $\mathcal{T} \in \mathcal{R}^{100 \times 100 \times 100}$ , which corresponds to the training set. As result an approximation  $\mathcal{T}'_{(a_1, a_2, a_3)}$  is retained. A further factor, which significantly determines the quality of the tag analysis, are the values set for  $a_1$ ,  $a_2$  and  $a_3$ . They determine the dimensions of the core tensor. Due to lack of time, only combinations with  $a_1 = a_2 = a_3$  were considered. A more detailed examination might be a task for future work.

In order to choose the setting of  $a_1$ ,  $a_2$  and  $a_3$ , which leads to the best recommendations based on the approximation  $\mathcal{T}'_{(a_1, a_2, a_3)}$ , we consider the precision for  $N = 1$ . It is assumed, that the higher this value is, the better the tag recommendations are. Of course this is a simplification. A more sophisticated approach to determine the factors, which indicate the quality of the tag analysis might be a future task.

For the random selection of the test set, setting  $a_1 = a_2 = a_3 = 10$  leads to the best tag recommendations. The resulting precision and recall for different values of  $N$  is plotted in the following coordinate system.

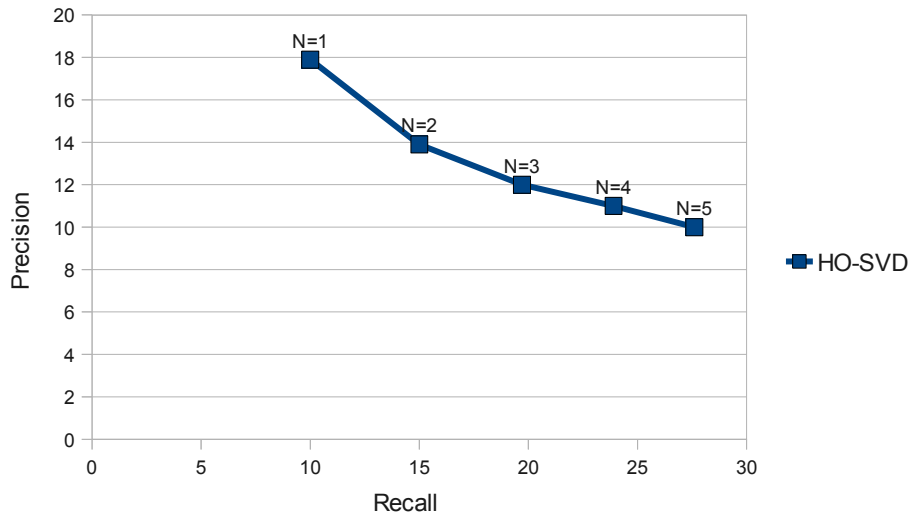


Figure 4.1: Results for random selection

Considering the more realistic scenario of temporal selection, even higher values for precision and recall are attained. This is a desirable property. Setting  $a_1 = a_2 = a_3 = 7$  leads to the best results.

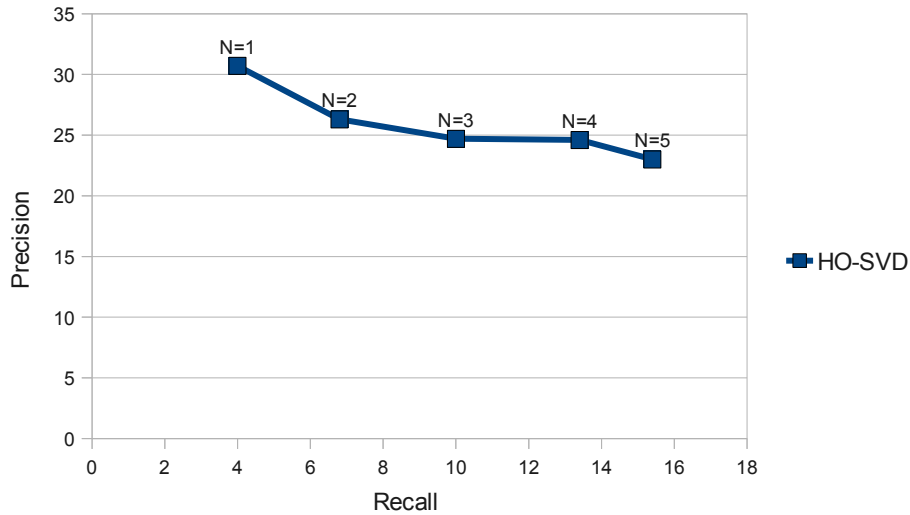


Figure 4.2: Results for temporal selection

In both scenarios the precision falls as  $N$  increases. In contrast to that the recall also increases as  $N$  increases. This behaviour was also observed in previous works, e.g. [23], and is quite intuitive. This leads us to the conclusion that the tag analysis qualitatively succeeded.

## 4.5 Comparison with a Two Dimensional Approach

However, as mentioned in [13], precision and recall aren't suitable to be considered as an absolute measure for the quality of tag recommendations. They should only be used in a comparative fashion on the same data set. Therefore another approach for computing the top- $N$  list of tags for a given user and picture was implemented and analysed.

This approach examines, whether the 3-order associations can be captured by a 2-dimensional approach using SVD. The method is introduced in [22]. First of all the training tensor  $\mathcal{T} \in \mathcal{R}^{100 \times 100 \times 100}$  is transformed into a tag-⟨user,picture⟩ matrix  $\mathcal{M} = \mathcal{T}_{(3)} \in \mathcal{R}^{100 \times 10000}$  using the 3rd-mode matrix unfolding of  $\mathcal{T}$ .

Then an approximation  $\mathcal{M}_k$  of  $\mathcal{M}$  is computed using SVD with respect to the  $k$  largest singular values. Similarly to the HO-SVD approach, the entries in the matrix are interpreted as probabilities. The column number  $c$ , which corresponds to user  $x$  and picture  $y$ , can be determined using the equation from chapter 2.2.1. Then the top- $N$  list of tags for a given user  $x$  and a picture  $y$  consists of the  $N$  tags with the highest values in column  $c$ .

In the scenario of the random selection the best results using the method with SVD are achieved for  $k = 8$ . However precision and recall are far below the level which is reached by an analysis with Higher-Order SVD.

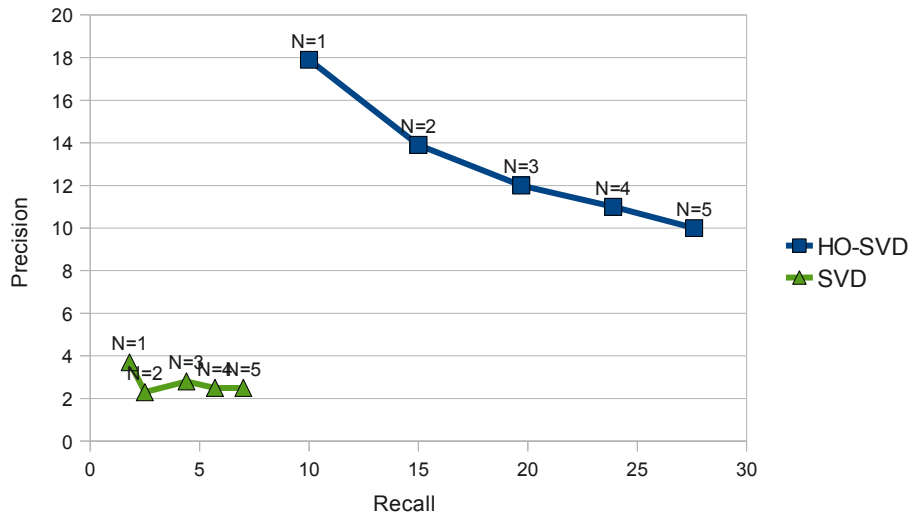


Figure 4.3: Comparison of results for random selection

For the temporal selection also  $k = 8$  yields the best results. Though the difference between the analysis with HO-SVD and the 2-dimensional approach with SVD isn't as big as in the random selection, it's still significant.

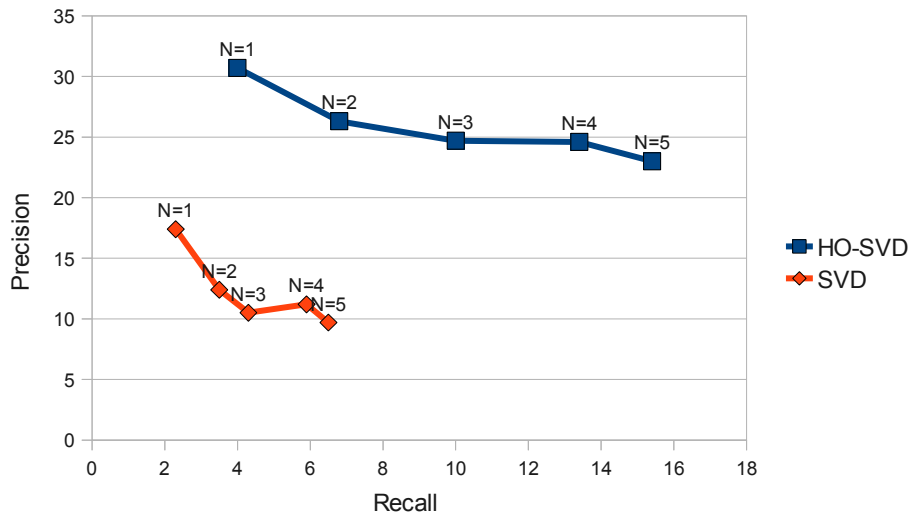


Figure 4.4: Comparison of results for temporal selection

As supposed, analysing the 3-dimensional associations with the 2-dimensional approach SVD isn't able to compete with the results gained by a higher-Order SVD. This underlines the importance of HO-SVD in the context of analysing multidimensional data.

## Chapter 5

# Conclusion

First of all this work introduces Latent Semantic Indexing (LSI) as motivation. This technique enables semantic search instead of full-text search in information retrieval systems. LSI analyses term-document matrices (i.e. 2-dimensional data) using Singular Value Decomposition (SVD). The mathematical background of SVD and its connection to LSI is explained.

However detecting, latent semantic structures in data, which depends on more than two factors, is a challenge in many applications, e.g. personalised web search. Higher-Order Singular Value Decomposition (HO-SVD) is an approach which addresses this issue.

Since a data structure for representing multidimensional data is needed, the concept of tensors is introduced. Then the mathematical basis of HO-SVD is explained. Finally a novel formalisation of a sequential algorithm for computing the HO-SVD of  $n$ th-order tensors is presented. It extends the existing CubeSVD algorithm and thus is called Extended CubeSVD.

Implementing this algorithm is a central part of this work. As basis an appropriate mathematical library had to be chosen. Even the library with the best support for tensor algebra needed to be extended in this project. The resulting implementation of Extended CubeSVD is used to analyse multidimensional data with Higher-Order SVD.

From an application called Artigo, 3-dimensional data from the context of social tagging was gathered. The idea is to measure the performance of tag recommendations and deduce the quality of the tag analysis from these results. By comparison to another approach, which transforms

the 3-dimensional data into a matrix and applies SVD, our implementation of HO-SVD attains significantly better results. This underlines the importance and the efficiency of HO-SVD for analysing multidimensional data. A future task would be a survey to compare the results of HO-SVD approaches to other sophisticated approaches like FolkRank or the Penalty-Reward algorithm.

Measuring the similarity of objects of the same kind is still an open question in the analysis of multidimensional data with HO-SVD [20]. However, this information is very important in many applications. Hence, examining this theme might be a central part of future work.

A further challenge of the HO-SVD approach is the rather high runtime complexity in combination with the typically high amount of data, that appears in many applications. This fact can lead to unreasonably high runtimes in practise. Hence efforts in order to speed up the algorithm, e.g. by parallelising it as mentioned in chapter 2.2.3, might be subject of further investigations on the one hand. On the other hand, efforts to reduce the size of the input data in advance (e.g. by clustering) may be promising.

# Bibliography

- [1] H. Arndt. UJMP Universal Java Matrix Package. <http://www.ujmp.org/>.
- [2] H. Arndt, M. Bundschuh, and A. Naegele. Towards a Next-Generation Matrix Library for Java. In *COMPSAC '09: Proceedings of the 2009 33rd Annual IEEE International Computer Software and Applications Conference*, pages 460–467, Washington, DC, USA, 2009. IEEE Computer Society.
- [3] R. Brent and F. Luk. The Solution of Singular-Value and Symmetric Eigenvalue Problems on Multiprocessor Arrays. *SIAM Journal on Scientific and Statistical Computing*, 6:69–84, 1985.
- [4] CERN European Organization for Nuclear Research. *Colt*. <http://acs.lbl.gov/software/colt/>.
- [5] L. De Lathauwer, B. De Moor, and J. Vandewalle. A Multilinear Singular Value Decomposition. *SIAM J. Matrix Anal. Appl.*, 21:1253–1278, 2000.
- [6] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science*, 41:391–407, 1990.
- [7] J. Demmel. *Templates for the Solution of Algebraic Eigenvalue Problems*, chapter Iterative Algorithms. Siam, 2002. online version: <http://www.cs.utk.edu/~dongarra/etemplates/index.html>.
- [8] J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart. LINPACK. <http://www.netlib.org/linpack/>.
- [9] J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart. *LINPACK User's Guide*. SIAM, 1979.

- [10] G. H. Golub and C. Reinsch. Singular Value Decomposition and Least Squares Solutions. *Numerische Mathematik*, 14:403–420, 1970. 10.1007/BF02163027.
- [11] G. H. Golub and C. F. van Loan. *Matrix Computations*. The Johns Hopkins University Press, 3rd edition, 1996.
- [12] G. Gundersen and T. Steihaug. Data Structures in Java for Matrix Computations: Research Articles. *Concurr. Comput. : Pract. Exper.*, 16(8):799–815, 2004.
- [13] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating Collaborative Filtering Recommender Systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, 2004.
- [14] L. Hogben. *Handbook of Linear Algebra*. Boca Raton CRC Press, 2007.
- [15] A. Hotho, R. Jäschke, C. Schmitz, and G. Stumme. Information Retrieval in Folksonomies: Search and Ranking. In Y. Sure and J. Domingue, editors, *The Semantic Web: Research and Applications*, volume 4011 of *Lecture Notes in Computer Science*, chapter 31, pages 411–426. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.
- [16] H. Kohle. Artigo 3.0 Social Image Tagging. <http://artigo.gwi.uni-muenchen.de/>.
- [17] B. Liu. *Web Data Mining*. Springer Berlin Heidelberg, 2nd edition, 2008.
- [18] The MathWorks and the National Institute of Standards and Technology (NIST). *JAMA: A Java Matrix Package*. <http://math.nist.gov/javanumerics/jama/>.
- [19] M. Odersky, P. Altherr, V. Cremet, I. Dragos, G. Dubochet, B. Emir, S. McDirmid, S. Micheloud, N. Mihaylov, M. Schinz, L. Spoon, E. Stenman, and M. Zenger. *An Overview of the Scala Programming Language (2. Edition)*. 2006.
- [20] P. Shah. Towards Efficient Algorithms for Higher-Order Singular Value Decomposition. 2005.
- [21] A. Singhal. Modern Information Retrieval: A Brief Overview. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 24(4):35–42, 2001.

- [22] J.-T. Sun, H.-J. Zeng, H. Liu, Y. Lu, and Z. Chen. CubeSVD: A Novel Approach to Personalized Web Search. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 382–390, New York, NY, USA, 2005. ACM Press.
- [23] P. Symeonidis, A. Nanopoulos, and Y. Manolopoulos. Tag recommendations Based on Tensor Dimensionality Reduction. In *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems*, pages 43–50, New York, NY, USA, 2008. ACM.
- [24] L. von Ahn and L. Dabbish. Labeling Images with a Computer Game. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 319–326, New York, NY, USA, 2004. ACM.
- [25] Z. Xu, Y. Fu, J. Mao, and D. Su. Towards the Semantic Web: Collaborative Tag Suggestions. In *WWW2006: Proceedings of the Collaborative Web Tagging Workshop*, Edinburgh, Scotland, 2006.
- [26] Y. Yang. Noise Reduction in a Statistical Approach to Text Categorization. In *SIGIR '95: Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 256–263, New York, NY, USA, 1995. ACM.