Technical University of Crete

Department of Electronic and Computer Engineering

Reviewer Profiling Using Factor Analysis

Submitted to the Department of Electronic and Computer Engineering in partial fulfillment of the requirements for the ECE Diploma Degree

By:
Evangelos E. Papalexakis

Advisor: Professor Nicholaos Sidiropoulos

Co-advisor: Professor Minos Garofalakis

Co-advisor: Professor Athanasios Liavas

Submitted: February 26, 2010

2

.

Abstract

Thousands of scientific conferences happen every year and each involves a laborious scientific peer review process conducted by one or more prominent scientists serving as Technical/Scientific Program Committee (TPC) chair(s). The chair(s) must match submitted papers to their reviewer pool in such a way that *i*) a paper is reviewed by experts in its subject matter; and *ii*) no reviewer is overloaded with reviews or under-utilized.

The paper to reviewer assignment requires prior knowledge about each reviewer's field of expertise and experience. This is essential because it produces reviewer assignments according to each reviewer's expertise. The TPC chair may do the assignment by inspection or by trial-and-error. For example, in a typical conference with a few hundred submitted papers, this process may involve a week of hard work from the chair.

The TPC chair may resort to reviewer and paper profiling, in order to categorize reviewers with respect to their field of expertise and research interests -regarding the reviewers- and addressed problem and specific scientific field - regarding the papers. When it comes to keywords that summarize a paper, most of the times a set of index terms is provided by the author. On the contrary, when it comes to reviewers, the TPC chair should extract the reviewers' profiles manually.

Reviewer profiling can be a frustrating and time consuming task; these properties often lead to sub-optimal assignments due to time pressure or insufficient knowledge of every reviewer's work. To that end, we propose two novel algorithms that approach the Reviewer profiling problem, in the context of a specific conference, in an automated manner. The formulation of these algorithms is based on Sparse Matrix Factorization and PARAFAC analysis. We also provide a comparison of the developed algorithms with a method widely used in Text Mining applications: Non-negative Matrix Factorization.

Acknowledgments

First and foremost, i would like to thank my family and dedicate them this work. If it had not been for their continuous and selfless support, the fulfillment of this thesis would not be possible.

I would of course like to thank my teacher and advisor, professor Nikos Sidiropoulos, for his valuable assistance, collaboration and guidance through every step of the present thesis and the shaping of my academic personality.

Finally, i would like to thank my good friends at the Technical University of Crete (you know who you are), for the wonderful experiences we shared the last 5 years. I love you all! Last but not least, i would like to thank my dear Theodosia, for putting up with me and making the last few years the best years of my life.

Contents

Preface

Notation and Symbols used

Notation

The end of an example is denoted by : $\square$

The acronym *iff* stands for *if and only if.*

A column vector is represented by a boldface, lowercase letter, e.g $\mathbf{v}$.

$\mathbf{a}^T\mathbf{b}$ denotes the inner product between the vectors $\mathbf{a}, \mathbf{b}$.

$\mathbf{a}\mathbf{b}^T$ denotes the outer product between the vectors $\mathbf{a}, \mathbf{b}$.

A matrix is represented by a boldface, uppercase letter, e.g $\mathbf{A}$.

For a $I \times J$ real valued matrix $\mathbf{A}$ we can say that $\mathbf{A} \in \mathbb{R}^{I \times J}$. Note that for $J = 1$, the same notation applies for a vector.

For an $I \times J$ complex valued matrix $\mathbf{A}$ we can say that $\mathbf{A} \in \mathbb{C}^{I \times J}$.

Element (i, j) $\mathbf{A}$ is denoted as $a_{i,j}$.

A Tensor or three-way array is represented by a calligraphic, uppercase letter, e.g $\mathcal{X}$.

Let us consider a $I \times J \times K$ tensor(or three-way array). We call a **slice** or **slab** the $I \times J$, $I \times K$ or $K \times J$ matrix that is produced by fixing one of the corresponding three indices. Such a slice of tensor $\mathcal{X}$ is denoted as $\mathbf{X}_k$.

$Pr\{\}$ denotes probability.

Unless stated otherwise, the convergence criterion for an optimization algorithm is defined as the difference of the cost function in two successive iterations to be less or equal to a small positive number.

Symbols used

$\mathbf{A}^T$: Matrix transposition operator

$\mathbf{A}^*$: Matrix Hermitian operator (conjugate transpose)

$\mathbf{A}^\dagger$: Matrix pseudo-inverse.

$\odot$: Hadamard product (elementwise matrix multiplication)

$\otimes$: Kronecker product.

$*$: Khatri-Rao product.

$\oslash$: Elementwise Matrix division.

$\mathbb{R}$: The set of real numbers.

$\mathbb{C}$: The set of complex numbers.

$\| \mathbf{A} \|_1 = \sum_{j=1}^{J} \sum_{i=1}^{I} |a_{i,j}|$: 1 norm of matrix $\mathbf{A}$ of size $I \times J$.

$\| \mathbf{A} \|_F = \sqrt{\sum_{i=1}^{I} \sum_{j=1}^{J} |a_{i,j}|^2}$: Frobenius norm of matrix $\mathbf{A}$ of size $I \times J$.

Preliminary Definitions

Definition 1 (Rank one matrix)**.** *A rank one matrix is defined as the outer product of two vectors.*

Definition 2 (Matrix rank)**.** *Let us consider a matrix* $\mathbf{M}$*. The minimum number of rank-one components (matrices) needed to decompose* $\mathbf{M}$ *is called the rank of* $\mathbf{M}$*.*

Definition 3 (Full rank matrix)**.** *An* $I \times J$ *matrix* $\mathbf{A}$ *is called* ***full-rank*** *if rank*$(\mathbf{A}) = min(I, J)$

Definition 4 (Hermitian Matrix)**.** *Consider a complex* $m \times m$ *matrix* $\mathbf{A} \in \mathbb{C}^{m \times m}$*.* $\mathbf{A}$ *is called* ***Hermitian*** *if*

$$\mathbf{A} = \mathbf{A}^* \tag{1}$$

Definition 5 (Positive Semidefinite Matrix)**.** *Consider an* $m \times m$ ***Hermitian*** *matrix* $\mathbf{A} \in \mathcal{C}^{m \times m}$ *with eigenvalues* $\{\lambda_k\}, k = 1 \cdots m$*.* $\mathbf{A}$ *is said to be* ***positive semidefinite*** *if any of the following equivalent conditions hold:*

1. $\lambda_k \geq 0, \forall k = 1 \cdots m$
2. $\mathbf{a}^* \mathbf{A} \mathbf{a} \geq 0, \forall \mathbf{a} \in \mathbb{C}^{m \times 1}$
3. $\exists$ *matrix* $\mathbf{C}$ *such that* $\mathbf{A} = \mathbf{C}\mathbf{C}^*$ *(*$\mathbf{C}$ *is called a square root of* $\mathbf{A}$*)*
4. *The determinant of any submatrix of* $\mathbf{A}$ *(obtained by striking out any rows and the corresponding columns) is* ≥ 0*.*

Definition 6 (Positive Definite Matrix)**.** *A* ***Hermitian*** *matrix* $\mathbf{A} \in \mathbb{C}^{m \times m}$ *is called* ***positive definite*** *if the conditions described for the positive semidefinite matrix hold with strict* > 0*.*

Corollary 1. *A matrix is reversible iff it is positive definite.*

List of Algorithms

List of Figures

Chapter 1

Introduction

Thousands of scientific conferences happen every year and each involves a laborious scientific peer review process conducted by one or more prominent scientists serving as Technical/Scientific Program Committee (TPC) chair(s). The chair(s) must match submitted papers to their reviewer pool in such a way that *i*) a paper is reviewed by experts in its subject matter; and *ii*) no reviewer is overloaded with reviews or under-utilized.

The paper to reviewer assignment requires prior knowledge about each reviewer's field of expertise and experience. This is essential because it produces reviewer assignments according to each reviewer's expertise. The TPC chair may do the assignment by inspection or by trial-and-error. For example, in a typical conference with a few hundred submitted papers, this process may involve a week of hard work from the chair.

Usually, the knowledge on each case is reflected on a small set of keywords that summarize the essence of the paper (when it comes to paper to session assignment) and define the reviewer's research profile. Towards the direction of keywords that summarize a paper, most of the times a set of index terms is provided by the author. On the contrary, when it comes to reviewers, no such analogy exists: A reviewer's profile needs to be extracted manually by the TPC chair either by using personal experience regarding the reviewer or by browsing through the list of his latest and/or highly cited publications. The latest publications usually provide a notion of his current research interests where the highly cited publications provide the backbone of his work. Both of the latter can be used by the TPC chair, in order to profile each reviewer.

Reviewer profiling proves to be a frustrating and time consuming task; these properties often lead to suboptimal assignments due to time pressure or insufficient knowledge of every reviewer's work. To that end, we propose two novel algorithms that approach the Reviewer profiling problem, in the context of a specific conference, in an automated manner. The formulation of these algorithms is based on Sparse Matrix Factorization and PARAFAC analysis.

Sparse Matrix Factorization is a subset of linear algebra that deals with the factorization of a sparse data matrix to sparse factors. Sparse Matrix Factorization can be seen also as a subset of Factor Analysis.

PARAFAC analysis is a subset of multi way analysis, a generalization of linear algebra for structures indexed by more than two indices, often called multi-way arrays or tensors. PARAFAC's roots can be traced in psychometrics and chemometrics. Some recent publications demonstrate PARAFAC analysis as a signal processing tool and as a Data Mining tool.

We also provide a comparison of the developed algorithms with a method widely used in Text Mining applications: Non-negative Matrix Factorization.

When describing the paper to reviewer and the paper to session assignment tasks, we placed the profiling process as a fundamental task that precedes the assignment. In the following lines we provide a brief description of the Reviewer Assignment Problem (RAP) and the Paper to Session Assignment problem, to get the reader acquintant with the whole picture the present thesis resides in.

1.1 The Reviewer-Assignment Problem

1.1.1 Definition

The Reviewer-Assignment problem is defined as follows: Consider a number of submitted publications to a conference or a journal. Let this number be n_p. Consider also a number of reviewers, n_r. Usually $n_r < n_p$. That means that one reviewer usually reviews more than 1 publications. The RAP is the problem of assigning every one of the n_p papers to each one of the n_r reviewers in a way that certain capacity and relevance/similarity constraints are met. The two types of constraints are:

- Capacity constraints: These constraints make sure that the assignment of the publications to each reviewer will be just, meaning that each reviewer should be assigned to review equal or approximately equal number of publications with everyone else, or an agreed upon quota. This group of constraints can be dealt with methods of optimization [8].

- Relevance constraints: These constraints make sure that this automatic assignment will take into account the field of expertise of each reviewer, assigning him the publications that are most correlated with his expertise. This group of constraints can be tackled by employing tools from fields of Natural Language Processing and Text Mining; This group of constraints is the heart of the present thesis.

We must note that apart from RAP, there exists the much similar problem of Paper-to-Session assignment. In this problem, instead of reviewers we have n_s session of a particular conference and we need to assign each paper to a session. In this case, the number of sessions is strictly less than the number of publications, (typically an order of magnitude less). The capacity constraints are obvious, since each session should contain equal or approximately equal number of publications. The relevance constraints ensure that papers in a session address related problems, therefore the session is coherent.

1.1.2 Related Work

Some recent work regarding the Reviewer Assignment and the Paper to Session assignment problems has been done in [8], where the problems are modeled and solved as optimization problems. A detailed survey about the Reviewer Assignment problem can be found in [9]. In the following lines, we provide an outline of the progress made on the RAP using data mining oriented approaches.

We review papers in the literature that deal with the assignment problem using data mining/information retrieval (IR) tools. The main concern among these papers is to compute the matching degree between papers and reviewers based on the content of the papers. Some of the related publications, like the present, attempt to recover each reviewer's biography through the internet, where other publications assume this information as given.

Apart from typical data mining/IR applications, that mainly use the Vector Space Model with the Latent Semantic Indexing expansion (principles that will be thoroughly discussed throughout this thesis), there exist publications that attempt to tackle the RAP using Collaborative Filtering and Semantic Web applications (to collect data representing researches' expertise). These approaches lie well beyond the scope of the present thesis. A list of all the publications mentioned above can be found in [9].

However, none of the related publications addresses the profiling problem. We attempt to solve this problem by employing tools, which are mostly used in signal processing (among others, e.g. Chemometrics). Moreover, we provide an extension to the traditional Non-negative Matrix Factorization paradigm (a concept that in a sense generalizes Latent Semantic Indexing) by introducing Sparse Matrix Regression.

1.2 Objective

This thesis is concerned with the extraction of reviewer profiles using information regarding the reviewers and information coming from the titles of the submitted publications that jointly represent the reviewers and the submitted papers. We use exclusively the title of each publication. Even though the full text is available to the TPC chair, we focus on the titles because they contain the distilled essence of the paper as noted and extracted by the author; when coming up with an appropriate title, an author does all the essential filtering of information we would normally do if we selected to focus on full texts and this filtering produces better results than any automated procedure. However, all the methods developed in the present thesis can be also applied to abstract or even full text indexing.

To state more formally the desired outputs, we introduce the following structures:

- A set of terms (moderately small in size) that best summarizes both the reviewers and the papers. Let us denote this set of terms as $\underline{\mathbf{T}}$ and it's cardinality as T_{final}.
- A binary matrix $\tilde{\mathbf{R}}$ of size $R \times T_{final}$ that is defined as follows:

$$\tilde{\mathbf{R}} \in [0,1]^{R \times T_{final}}$$

$$\tilde{r}_{i,j} = \begin{cases} 1 & \text{if term } j \text{ matches reviewer } i \\ 0 & \text{otherwise} \end{cases}$$
$$i = 1 \cdots R, j = 1 \cdots T_{final}$$

  where R is the number of reviewers.
- A binary matrix $\tilde{\mathbf{P}}$ of size $P \times T_{final}$ that is defined in a similar way as $\tilde{\mathbf{R}}$

$$\tilde{\mathbf{P}} \in [0,1]^{P \times T_{final}}$$

$$\tilde{p}_{i,j} = \begin{cases} 1 & \text{if term } j \text{ matches paper } i \\ 0 & \text{otherwise} \end{cases}$$
$$i = 1 \cdots P, j = 1 \cdots T_{final}$$

  where P is the number of the submitted papers.

1.3 Thesis Outline

The outline of this this is as follows: In Chapter 1 we discuss the Reviewer-Assignment problem and the related work existing in the literature. In Chapter 2 we provide a short introduction to Text Mining, describing the theory and tools used as the backbone of this work. This chapter also goes on and discusses the process of Text Mining from the World Wide Web and how it is connected to our work. In chapters 3-5 we provide the essential mathematical background ,more specifically: in Chapter 3 we introduce the concept of Matrix Factorization (the bilinear decomposition) and methods to achieve this, in Chapter 4 we introduce Sparse Matrix Regression as an alternative to the Matrix Factorization and in Chapter 5 we introduce multiway analysis and the PARAFAC decomposition. In chapters 6 & 7, we provide the algorithms we developed and the results/conclusion of our work.

Finally, some useful results and conclusions are included in Appendices A & B. The results obtained there are excluded from chapter 7 due to their supplementary nature, however, they offer a qualitative evaluation based on real world data.

Chapter 2

Text Mining

Text Mining is the field of Information Retrieval or Data Mining that refers to the process of discovering useful information and emerging patterns from text. Text mining usually involves the process of structuring and parsing the input text , deriving patterns within the structured data, and finally evaluation and interpretation of the output. Typical text mining tasks include *text clustering*, *query evaluation*, *concept extraction* and *term extraction*.

This thesis focuses on the concept extraction aspect of text mining, as our main objective is to extract representative term profiles for a set of submitted papers to a conference and a set of reviewers to the same conference. In the following sections, we provide the theoretical background behind the foundations of text mining used along this thesis. More specifically, we describe the Vector Space Model of Information Retrieval, the extraction of multi-word terms and an automated tool that employs machine learning methods in order to extract terms from a document or a collection of documents.

2.1 Theoretical Background

2.1.1 The Vector Space Model

Introduction

Vector space model is called the algebraic model used for the representation of text documents as vectors in a space defined by a set of index terms (also found as *dictionary* in the literature) as a basis vector. The Vector Space Model is widely used since the 1960s when it was first proposed, due to it's elegance and precise algebraic formulation.

Index Terms

As mentioned above, the Vector Space Model (VSM) creates a mapping of a document to a vector space defined by the set of index terms. Usually, when having a collection of documents, e.g the set of all the submitted papers to a conference, the index terms can be defined as the set of all the terms found throughout all the documents (or the titles, whether the indexing is based upon the whole text or just the title, which happens to be our case), excluding the so-called *stop-words*. A stop-word can be defined as a term that is either too frequent-typical (thus proving to be rather useless) or a preposition, pronoun, conjunction or interjection etc. From now on, when referring to a term, we refer to a single or multi word term. In addition, we usually use the stem of each term in order to avoid essentially duplicate terms, for example give/giving or publication/publications etc.

From Document to Vector

The depiction of a document to the vector space defined by the dictionary basis vector is very simple and elegant. Let us consider the $I \times 1$ vector $\mathbf{d}_i$ representing the i-th document of the set $\mathcal{D}$ of all

the documents contained in the collection(often called **corpus** in the literature). The vector $\mathbf{d}_i$ can be defined as:

$$\mathbf{d}_i = \begin{cases} w_{i,j} & \text{if the term } j \text{ exists in the document} \\ 0 & \text{else} \end{cases} \tag{2.1}$$
$$i = 1 \cdots N, \text{ where N is the cardinality of the set } \mathcal{D}$$

The number $w_{i,j}$ denotes the weight of the term j in the document i. There are several approaches to the definition of that weight, depending on the application itself.

Term Weighting

The weight $w_{i,j}$ can be the frequency of occurrence of the j-th term inside the i-th document. Even though raw term frequency makes intuitive sense, it suffers from a critical problem: All terms are considered equally important when it comes to assessing relevancy on a query. In fact, certain terms have little or no discriminating power in determining relevance. For instance, a collection of documents on the auto industry is likely to have the term auto in almost every document. To this end, we introduce a mechanism for attenuating the effect of terms that occur too often in the collection to be meaningful for relevance determination. An immediate idea is to scale down the term weights of terms with high collection frequency, defined to be the total number of occurrences of a term in the collection. The idea is to reduce the term frequency weight of a term by a factor that grows with its frequency throughout the corpus.

Instead, it is more commonplace to use for this purpose the document frequency df , defined to be the number of documents in the corpus that contain a term t. This is because in trying to discriminate between documents for the purpose of scoring, it is better to use a document-level statistic (such as the number of documents containing a term) than to use a collection-wide statistic for the term.

How is the document frequency df of a term used to scale its weight? To that end, we need to define the inverse document frequency (idf) of the j-th term t_j as follows:

$$\text{idf}_j = \log \frac{N}{\text{card}(t_j \in \mathcal{D})} \tag{2.2}$$

where $\text{card}(t_j \in \mathcal{D})$ denotes the number of documents of the collection $\mathcal{D}$ that contain the term t_j. We also define as $\text{tf}_{i,j}$ the frequency of the j-th term occurring in the i-th document.

The weight of the i-th document for the j-th term now becomes:

$$w_{i,j} = \text{tf}_{i,j}\text{idf}_j \tag{2.3}$$

introducing the **term-frequency/inverse document frequency** weighting scheme.

Applications of VSM

The most popular application of the VSM in text mining and information retrieval is it's use for query relevance evaluation and document clustering.

Query Evaluation Let us consider the document corpus $\mathcal{D}$ and a query upon the contents of that corpus. Such a query usually consists of 1-2 sentences and expresses a question in natural language. VSM provides us with a way to determine the relevance of every document contained in $\mathcal{D}$ to a query. We can represent a query the same way we represent a document in the vector space defined by our index terms. The term weighting of the query vector can be binary, indicating whether a term appears in the query or not. The similarity of each document to the query can be computed using the cosine of

the angle between the two corresponding vector. This is usually referred to as **cosine similarity** and can be computed by the inner product of the two vectors as follows:

$$\text{sim}(\mathbf{d}_i, \mathbf{q}) = \cos(\theta_{\mathbf{d}_i,\mathbf{q}}) = \frac{\mathbf{d}_i^T \mathbf{q}}{\| \mathbf{d}_i \| \| \mathbf{q} \|} \tag{2.4}$$

where $\mathbf{q}$ is the query vector, $\mathbf{d}_i$ is the i-th document vector and $\theta_{\mathbf{d}_i,\mathbf{q}}$ is the angle between the former two vectors. After computing the similarity of each document to the query, the documents can be sorted according to their similarity to the query, usually referred to as score or ranking.

Document Clustering The task of document organization in categories is called Document Clustering. The categorization relies on a predefined similarity criterion. The similarity criterion that relates to the VSM is the cosine similarity criterion that is defined in equation 2.4. Instead of calculating the similarity of a document to a query, in the clustering concept, we calculate the document to document similarity and cluster the documents of the corpus $\mathcal{D}$ accordingly.

Term/Concept Extraction Term/Concept Extraction is the application of interest in this thesis. Its aim is to extract appropriate and representative terms or group of terms from large amounts of text.

2.1.2 Multi-word term extraction

Multi-word terms extraction does not differ in principle from single-word term extraction. The term-by-document matrix could be created using a basis of both single and multi word terms. However, when in need to extract multiword terms, one should consider a way of determining whether a certain combination of word that form a multi word term is meaningful or not. To this end, we present a practical method for the extraction of multi word terms, first proposed in [10]. In [10], two methods called C-value and NC-value regarding the weighting of multi word terms are introduced. We shall present these two methods briefly in this section:

- **C-value**: The C-value is a metric that measures the significance of a multi-word term as an actual term by measuring it's occurrence in sequences of text. If the length of the multi-word term is maximal, it weight is obtained by it's frequency of occurrence or any other alternative weighting scheme that we demonstrated previously. However, if the candidate term is a subsequence of some longer candidate term, then it is possible that the longer term is a more appropriate term than it's subsequence, thus the C-value weighs the candidate term that appears as a subsequence to longer terms, according to the weights of the corresponding longer terms. Other than this short introduction, the C-value is beyond the scope of this thesis, but the reader can find a more detailed description in [10].
- **NC-value**:The NC-value, additionally to the C-value, attempts to incorporate information regarding the context of each candidate term. Thus, the main idea is that if the context of a candidate multi-word term is highly weighted, this would probably lead to a good candidate mutli-word term. When referring to the context of a candidate term, we refer to the terms surrounding that candidate term. The NC-value of a term 'a' can be stated by the following formula:

  $$\text{NC-value}(a) = 0.8\text{C-value}(a) + 0.2 \sum_{b \in C_a} f_a(b)\text{weight}(b)$$

  where C_a is the set of context terms of candidate term 'a', $f_a(b)$ is the frequency of term 'b' as a context of 'a' and weight(b) is the weight of 'b' as a term context word.

2.2 The KEA Algorithm and Tool

Instead of parsing/pre-processing the text ourselves, creating the term-by-document matrix, choosing the appropriate weighting scheme and proceeding manually to the term extraction process, we can choose

to employ certain tools that excel in that field. One of these tools, that we use in this thesis, is the KEA tool. KEA is an abbreviation for *Keyphrase*[1] *Extraction Algorithm* and is introduced in [11].

KEA's extraction algorithm consists of the two following stages

- Training: Creation of a model for identifying keyphrases using training documents where the author's keyphrases are known. For the purposes of the training process, KEA uses the **Naive Bayes** machine learning technique.

- Extraction: In this step, the set of keyphrases to be extracted from a new document is done, based on the model produced from the Training step. For each candidate keyphrase, two features are computed: The TF×IDF weigh (as described previously) and the *first occurrence*, which is the distance into the document of the phrase's first appearance. Based on these two features, the Naive Bayes model is employed to determine whether this candidate phrase is indeed a proper term and should be output.

The KEA tool can be downloaded from the New Zealand Digital Library project (`http://www.nzdl.org/`)

2.3 Mining from the Web

The World Wide Web is a vast source of information. Most of this information basically exists in textual form, where other popular forms include images, videos and other type of binary files. The specific sub-field of Data Mining, Web Mining, is associated with the retrieval, indexing and extraction of hidden information from resources existing on the World Wide Web or a subset. The task of exploring the web, retrieving useful information out of it and further processing that information is fundamental to many every-day use applications such as web search engines (i.e Google). We are going to provide insight to the basics of that task, as far as our application is concerned. Throughout this section, we are about to describe the procedure of exploring the web and retrieving desired information. After the retrieval process, the indexing and processing process adheres to the principles of Text Mining that was demonstrated in chapter 2.

2.3.1 Exploring the Web

The task of exploring the web is quite appropriately called *crawling* and the programs that carry out that task are called in the same spirit *crawlers*, *bots* or *spiders*. The main task of such a program is quite simple; it navigates through a web page, downloads any contained information, discovers any existing hyperlinks to other web pages(on the same or different domain) and then navigates to each one of those hyperlinked pages, repeating the same pattern. One can notice that what a crawler does, does not differ in principle from what a regular human user does, only the crawler does that in a systematic pattern by following all the links and downloading every piece of downloadable information. The stopping criterion for a crawler is rather abstract and could be either when a specific subset of the web is covered, or when a time limit is reached and so on.

[1] In this context, 'keyphrase' has the same meaning with 'term'

Algorithm 2.1 A simple web crawler

Input: A list of starting web pages' addresses

```
1: Insert each of the starting addresses to a structure S. S can either be a LIFO or a FIFO structure.
2: while stoping criterion is not met do
3:    Pick a page p from S.
4:    Download the content of that page.
5:    Process the content in order to discover any hyperlinks.
6:    for all hyperlinks in p do
7:       insert hyperlink into S
8:    end for
9: end while
```

On algorithm 2.1 is stated that the structure **S** where the crawler deposits the candidate links can be either a FIFO (First In First Out) or a LIFO (Last In First Out) structure. The difference between the use of FIFO or LIFO is the fact that the exploration using a FIFO structure would lead to a 'deeper' search in the web graph where using a LIFO structure would lead to a 'broader' search. However, further discussion upon this fact is beyond the scope of this introduction.

The step of crawling is one of the most important tasks of the whole web mining process. It is not exaggeration to say that every known web search engine does that in a very regular basis, updating its archives of indexed web pages (or what many times might resemble a replica of the World Wide Web).

Chapter 3

Matrix Factorization

3.1 Introduction

This chapter focuses on the factorization of a matrix $\mathbf{M}$ in 2 factors $\mathbf{A}, \mathbf{B}$, such that:

$$\mathbf{M} = \mathbf{A}\mathbf{B}^T \tag{3.1}$$

Usually, this factorization is done on a lower rank than the original rank of matrix $\mathbf{M}$, so the above equation becomes an approximation:

$$\mathbf{M} \approx \mathbf{A}_k \mathbf{B}_k^T \tag{3.2}$$

Where k is the inner dimension of the two factors and also the approximation rank.

By factoring $\mathbf{M}$ in the above form, we can express each row of the original matrix as a linear combination of the columns of $\mathbf{B}$, weighed by the rows of $\mathbf{A}$ in a low-dimensional latent space.

This type of factorization can be achieved by employing either one of the following techniques. In the following sections, we describe the principles governing the *Singular Value Decomposition* and the *Non-negative Matrix Factorization* and provide the methods for obtaining the factors $\mathbf{A}, \mathbf{B}$ for each one.

3.1.1 Definition of Matrix Rank

Consider an $I \times J$ matrix $\mathbf{M}$, and suppose that rank($\mathbf{M}$) = 3. Let $m_{i,j}$ denote the (i,j)-th entry of $\mathbf{M}$. Then it holds that $m_{i,j}$ admits a *three-component bilinear decomposition*:

$$m_{i,j} = \sum_{f=1}^{3} a_{i,f} b_{j,f} \tag{3.3}$$

for all $i = 1 \cdots m$ and $j = 1 \cdots n$. Equivalently, letting $\mathbf{a}_f = \begin{bmatrix} a_{1,f} & \cdots a_{I,f} \end{bmatrix}^T$ and $\mathbf{b}_f = \begin{bmatrix} b_{1,f} & \cdots a_{J,f} \end{bmatrix}^T$

$$\mathbf{M} = \mathbf{a_1}\mathbf{b_1}^T + \mathbf{a_2}\mathbf{b_2}^T + \mathbf{a_3}\mathbf{b_3}^T \tag{3.4}$$

i.e.,$\mathbf{M}$ can be written as a sum of three outer products of the respective component vectors, which constitute rank-one matrices. The above decomposition can be found in figure 3.1 below:

According to the above, we can derive the following definition for matrix rank:

Definition 7 (Matrix Rank)**.** *Let us consider a matrix* $\mathbf{M}$*. The minimum number of rank-one components (matrices) needed to decompose* $\mathbf{M}$ *is called the rank of* $\mathbf{M}$*.*

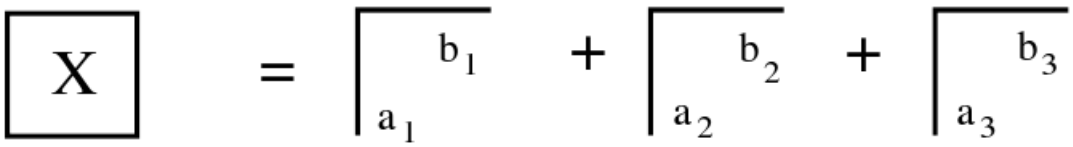


Figure 3.1: The Matrix Bilinear Decomposition. (Copyright belongs to [1])

3.2 Singular Value Decomposition (SVD)

3.2.1 Definition and Properties

Definition 8 (Unitary Matrix)**.** *Let us consider a complex $I \times J$ matrix $\mathbf{U}$. $\mathbf{U}$ is called unitary if*

$$\mathbf{U}\mathbf{U}^* = \mathbf{U}^*\mathbf{U} = \mathbf{I}_n \tag{3.5}$$

Corollary 2. *If $\mathbf{U}$ is unitary then :*

$$\mathbf{U}^* = \mathbf{U}^{-1} \tag{3.6}$$

Theorem 1 (Singular Value Decomposition)**.** *Let us consider the $I \times J$ complex-valued matrix $\mathbf{M}$. Then there exists a factorization of the form:*

$$\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* \tag{3.7}$$

- $\mathbf{U}$ *Matrix of left singular vectors of $\mathbf{M}$. This matrix is $I \times I$ and unitary.*
- $\mathbf{V}$ *Matrix of right singular vectors of $\mathbf{M}$. This matrix is $J \times J$ and unitary.*
- $\mathbf{\Sigma}$ *Diagonal matrix that contains the singular values of $\mathbf{M}$. We may assume that the singular values are in a descending order. The diagonal elements of $\mathbf{\Sigma}$ are non-negative.*

If matrix $\mathbf{M}$ is real-valued, the Singular Value Decomposition of $\mathbf{M}$ is:

$$\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{3.8}$$

*and all of the above properties hold. In that case of course, matrices $\mathbf{U}, \mathbf{V}$ are said to be **orthogonal**, an equivalent property of a Hermitian matrix when it comes to real valued matrices.*

Remark 1. *The two following points emerge as a result of the theorem 1*

- *The left singular vectors are also the eigenvectors of the matrix $\mathbf{A}\mathbf{A}^*$ (or $\mathbf{A}\mathbf{A}^T$ for the real case).*
- *The right singular vectors are also the eigenvectors of the matrix $\mathbf{A}^*\mathbf{A}$(or $\mathbf{A}^T\mathbf{A}$ for the real case).*

Corollary 3 (Compact SVD)**.** *Let us consider the $I \times J$ matrix $\mathbf{M}$ and $r = rank(M) = min(I, J)$. Then, the Singular Value Decomposition of $\mathbf{M}$ can be written as:*

$$\mathbf{M} = \mathbf{U}_r\mathbf{\Sigma}_r\mathbf{V}_r^* \tag{3.9}$$

where:

- $\mathbf{U}_r$ *$m \times r$ matrix containing the r first left singular vectors.*
- $\mathbf{V}_r$ *$n \times r$ matrix containing the r first right singular vectors.*
- $\mathbf{\Sigma}_r$ *$r \times r$ matrix containing all the singular values of $\mathbf{M}$.*

It can be proved that through the Singular Value Decomposition of a matrix, we can obtain the optimal low rank approximation of that matrix, in terms of the mean square error. This is summarized by the following theorem:

Theorem 2 (SVD approximation)**.** *Let us consider the rank r $I \times J$ matrix* $\mathbf{M}$ *and let us consider an integer $k < r$ being the desired approximating low rank. The optimal rank k approximation of* $\mathbf{M}$ *in the least squares sense is obtained through the truncated Singular Value Decomposition of* $\mathbf{M}$*:*

$$\mathbf{M}_k \approx \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^* \tag{3.10}$$

where

$\mathbf{U}_k$ *$m \times k$ matrix containing the k first left singular vectors.*

$\mathbf{V}_k$ *$n \times k$ matrix containing the k first right singular vectors.*

$\mathbf{\Sigma}_k$ *$k \times k$ matrix containing the k first singular values of* $\mathbf{M}$*.*

3.2.2 Matrix Factorization using the SVD

In order to factor $\mathbf{M}$ as the product of $\mathbf{A}_{svd}, \mathbf{B}_{svd}$ we can use the SVD approximation Theorem (2).

We take the svd approximation of $\mathbf{M}$ as presented in Theorem 2. Then we may define the 2 factors as:

$$\mathbf{A} = \mathbf{U}_k \mathbf{\Sigma}_k$$

$$\mathbf{B} = \mathbf{V}_k$$

3.2.3 Applictions of the SVD

Latent Semantic Indexing

One very useful application of the Singular Value Decomposition, perhaps the most fundamental in text mining is *Latent Semantic Indexing*, LSI for brevity which was first introduced in 1990 in [12].

Reconsider the Vector Space Model document representation, where each document of a collection $\mathcal{D}$ is represented by a I-dimensional vector $\mathbf{d}_n$, where I is the number of index term that constitute the basis vector. If the number of documents in $\mathcal{D}$ is N, then we may arrange all the document vectors in a $I \times N$ matrix $\mathbf{C}$ where the n-th column is $\mathbf{d_n}$.

Now, consider the low rank approximation of the term-by-document matrix $\mathbf{C}_k$ (as defined in Theorem 2), where k is integer and $k < \text{rank}(\mathbf{C})$. The principle of LSI states the substitution of $\mathbf{C}$ with $\mathbf{C}_k$ and use the approximation for carrying out queries. In VSM, queries are represented as I-dimensional vectors too. That being said, the need rises for the projection of the query vector $\mathbf{q}$ to the low-rank space of $\mathbf{C}_k$. This projection is done as follows:

$$\mathbf{q}_k = \mathbf{\Sigma}_k^{-1} \mathbf{U^T}_k \mathbf{q}$$

where $\mathbf{\Sigma}_k, \mathbf{U}_k$ are obtained by the svd approximation Theorem 2. Now, we may use cosine similarities to compute the similarity between a query and a document, between two documents, or between two terms.

Latent Semantic Index is shown to deal with the concepts of *polysemy* and *synonymy*, where VSM fails to. *Polysemy* refers to the case where a term has multiple meanings and *synonymy* refers to the case where two or more words have the same meaning. This can be justified by the dimensionality reduction provided by the low rank approximation. Namely, the rows of the term-by-document matrix are mapped to a lower dimension space, leading to the compaction of terms that behave the same way(thus not providing any new information in the original term-by-document matrix).

Link analysis (HITS algorithm)

The HITS algorithm (which is abbreviation for Hyperlink-Induced Topic Search) is a link analysis algorithm proposed by Jon Kleinberg in [13]. Given a directed graph $G = \{V, E\}$ of web pages where the vertices V represent the set of web pages and the edges E represent the hyperlinks that contain a reference from one page to the other, HITS computes a ranking of each page as a **hub** and as an **authority**. The definition of the two latter terms is simple: A *hub* is a page that contains many links to pages that have good content. The pages that are being referenced by many hubs and have high value of content are called *authorities*.

The algorithm proposed in [13] is a simple iterative algorithm that updates the hub and authority values for each page until a predefined convergence criterion is met. On the other hand, in [14] there is shown an alternative way of computing the hub and authority scores via the Singular Value Decomposition. In order to present this approach, first we must get the reader acquintant with the definition of the **Adjacency Matrix**, a way of representing a graph:

Definition 9 (Graph Adjacency Matrix)**.** *Let us consider a graph $G = \{V, E\}$. Without loss of generality, we may assume that this graph is directed, but the same definition applies for indirected graphs too. An Adjacency Matrix $\mathbf{A} \in [0, 1]^{|V|\times|V|}$ is defined as follows:*

$$a_{i,j} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{else} \end{cases} \tag{3.11}$$

If the graph is weighted then $\mathbf{A} \in \mathbb{R}^{|V|\times|V|}$ and instead of 1, each element contains the weight of the corresponding edge.

In [14] is shown that the hub scores of a set of webpages depicted by a matrix $\mathbf{A}$ can be obtained from the first left singular vector, while the authority scores can be obtained from the first right singular vector. Let us examine the following example, taken from [14] where the above property of SVD is illustrated:

Example 1. *Consider the graph in figure 1 that contains 4 web pages and the hyperlinks among them.*

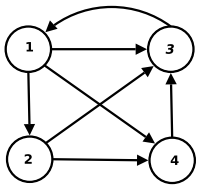


Figure 3.2: Graph representing 4 web pages and the hyperlinks among them

The Adjacency matrix that represents the above graph is

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \tag{3.12}$$

The first left singular vector is

$$\mathbf{u}_1 = \begin{bmatrix} 0.74 \\ 0.59 \\ 0.00 \\ 0.33 \end{bmatrix} \tag{3.13}$$

The above vector indicates that the webpage with the highest hub score is the first one. Accordingly, the first right singular vector is

$$\mathbf{v}_1 = \begin{bmatrix} 0.00 \\ 0.33 \\ 0.74 \\ 0.59 \end{bmatrix} \tag{3.14}$$

which indicates that the page with the highest authority score is the third.

□

Principal Component Analysis

Principal Component Analysis or PCA for brevity, is the workhorse of multivariate analysis in statistics, which provides a transformation of correlated data into a generally smaller coordinate space. This transformation is achieved either by the eigendecomposition or by the Singular Value Decomposition.

When using the SVD, Principal Component Analysis uses the same matrices that we introduce in the Matrix Factorization concept. To be more precise, the matrix holding the principal components is $\mathbf{A} = \mathbf{U\Sigma}$ and the matrix holding the loading values is $\mathbf{B} = \mathbf{V}^T$.

The image below, taken from [2], visualizes the above remark.

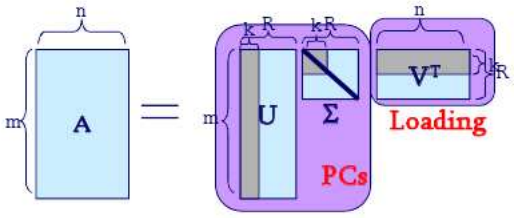


Figure 3.3: PCA using the SVD (Copyright belongs to [2])

Image Compression

This application relies on theorem 2 The idea behind the compression is the computation of a lower-rank approximation of the matrix that represents the digital image we want to compress. Of course, this application is merely academic than practical in terms of execution cost, but provides a very good visualization of theorem 2.

Example 2. *In this point we shall demonstrate the process of Image Compression via the SVD for a number of different rank approximations of the original image matrix. To that end, we use the widely used image of Lenna and exhibit a series of compressed images using different number of ranks(i.e different number of singular values on the truncated SVD(2)). The original image is*

The subscript under every image indicates the number of components used(thus the rank of the resulting matrix) in order to approximate the initial image. Note that the rank of the initial image is 254. We can see that even with a rank close to 50, we can approximate the image in acceptable quality.

□

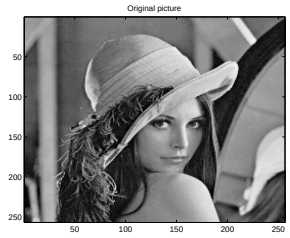
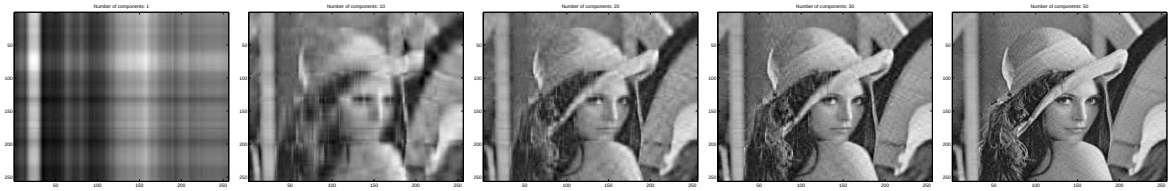


Figure 3.4: Original Image (rank=254)



(a) k=1 (b) k=10 (c) k=20 (d) k=30 (e) k=50

Figure 3.5: Image Compression via SVD approximation using k components

3.3 Non-Negative Matrix Factorization (NMF)

3.3.1 Definition

Non-negative Matrix Factorization can be expressed as a non-linear, constrained optimization problem, as follows: Consider a (generally) non-negative matrix $\mathbf{M} \in \mathbb{R}^{I \times J}$. We want to decompose $\mathbf{M}$ into two factors $\mathbf{A} \in \mathbb{R}^{I \times \hat{k}}$,$\mathbf{B} \in \mathbb{R}^{J \times \hat{k}}$, where $\hat{k} < \text{rank}(\mathbf{M})$ minimizing the following objective function:

$$\min_{\mathbf{A},\mathbf{B}} \| \mathbf{M} - \mathbf{A}\mathbf{B}^T \|_F^2 \tag{3.15}$$
$$\text{subject to } a_{i,j} \geq 0 \text{ and } b_{i,j} \geq 0$$

In [3], Non-negative Matrix Factorization is demonstrated for it's use as a sum-by-parts representation of image data in order to both identify and classify image features.

In this factorization principle, each row of the initial data matrix $\mathbf{M}$ is expressed as a linear combination of the columns of $\mathbf{B}$, weighted by the elements of the corresponding row of $\mathbf{A}$. The fact that this linear combination contains strictly additive relations between it's elements, provides a more interpretable model in contrast to the factorization using the SVD (presented in the previous section), that is compatible with human perception of a parts-based representation. Intuitively, a parts-based representation involves additive relationships between the parts that constitute the whole. Such representation is achieved through Non-negative Matrix Factorization.

3.3.2 Algorithms

Several algorithms have been proposed for the computation of the Non-negative Matrix Factorization. Most of the existent algorithms are described in [15].

Multiplicative Method

The most popular approach is the one proposed in [3], called *Multiplicative Method* and can be written in pseudocode as follows:

Algorithm 3.1 Non-negative Matrix Factorization using the Multiplicative Method

Input: $\mathbf{M}$ of size $I \times J$, $\hat{k}$
Output: $\mathbf{A}, \mathbf{B}$
  $\mathbf{A}$=randn(I,$\hat{k}$);
  $\mathbf{B}$=randn($\hat{k}$,J);
  while convergence criterion is not met **do**
    $\mathbf{B}$=$\mathbf{B}\odot(\mathbf{A}^T\mathbf{M})\oslash(\mathbf{A}^T\mathbf{A}\mathbf{B}+\epsilon)$;
    $\mathbf{A}$=$\mathbf{A}\odot(\mathbf{M}\mathbf{B}^T)\oslash(\mathbf{A}\mathbf{B}\mathbf{B}^T+\epsilon)$;
  end while
  $\mathbf{B}$=$\mathbf{B}^T$;

In algorithm 3.1, ϵ is a very small positive number used to avoid division by zero. Instead of initializing $\mathbf{A}, \mathbf{B}$ with totally random values, one can employ more sophisticated methods, such as using the Singular Value Decomposition of $\mathbf{M}$ to obtain initial values for the two factors [16].

The optimization problem 3.15 is non-convex, thus there exist more than one stationary points. Given that fact, convergence issues arise. Regarding the convergence of the Multiplicative Method we can read at [15] that when the Multiplicative Method algorithm converges to a limit point in the interior of the feasible region, the point is a stationary point. The stationary point may or may not be a local minimum. If the limit point lies on the boundary of the feasible region, one cannot determine its stationarity.

The computational complexity per iteration for Algorithm 3.1 is $\mathcal{O}(IJ\hat{k})$.

Alternating Least Squares

Another, more simplistic, approach to solving 3.15 is by decomposing the non-linear optimization problem into two linear optimization problems. This technique is called *Alternating Least Squares* in general and consists of the following simple steps:

- Fix matrix $\mathbf{A}$ to initially a random value(the same initialization issues as above apply here too).
- Solve the Non-negative Least Squares problem (NNLS) :

$$\min_{\mathbf{B}} \| \mathbf{M} - \mathbf{A}\mathbf{B}^T \|_F^2 \quad \text{subject to } b_{i,j} \geq 0$$

- Fix $\mathbf{B}$ to the value obtained at the previous step.
- Solve the Non-negative Least Squares problem (NNLS) :

$$\min_{\mathbf{A}} \| \mathbf{M} - \mathbf{A}\mathbf{B}^T \|_F^2 \quad \text{subject to } a_{i,j} \geq 0$$

- Repeat from the top until convergence.

3.3.3 Applications of NMF

Non-negative Matrix Factorization has a variety of applications, spanning from Image Classification [3], document clustering [17, 3], anomaly detection[18] and blind source separation[19] among others. In the following lines we will present two representative applications of NMF found in the literature.

Image Classification

In [3], NMF is used in an image classification context and is compared to Vector Quantization(VQ) and Principal Component Analysis(PCA). These three methods are applied on a database of facial images and the term of comparison is the way of representation of an image. All three methods represent an image as a linear combination of basis images. VQ and PCA representations allow the coefficients of the combination to be either additive or subtractive, whereas NMF permits only additive linear combinations. In [3] is demonstrated that this strictly additive representation leads to more intuitive data modelling. In figure 3.6, taken from [3], this difference between the three methods is demonstrated.

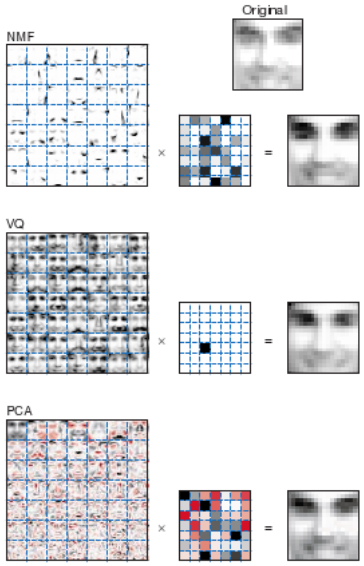


Figure 3.6: Image Classification using (a)NMF (b)VQ and (c)PCA (Copyright belongs to [3])

3.4 Identifiability

In this section, we examine the issue of the uniqueness of the bilinear decomposition. As it will follow, this model is generally not unique, but when some conditions hold, the NMF pertains uniqueness.

3.4.1 Rotational Freedom

A matrix factorization in the form

$$\mathbf{M} = \mathbf{A}\mathbf{B}^T$$

is also called a bilinear decomposition or bilinear model as it can also be expressed by the k component bilinear decomposition(where k is the low approximating rank):

$$\mathbf{M} = \mathbf{a}_1\mathbf{b_1}^T + \cdots + \mathbf{a}_k\mathbf{b_k}^T$$

It can be shown that the above model suffers from rotational indeterminacy, thus is not unique, even when posing non-negativity conditions in the general case. Let us consider the following example:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 2 & 3 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 3 & 3 \\ 4 & 5 \end{bmatrix}$$
$$\mathbf{M} = \mathbf{A}\mathbf{B}^T = \begin{bmatrix} 6 & 9 \\ 15 & 23 \end{bmatrix}$$

We can easily see that given $\mathbf{A}, \mathbf{B}$, $\mathbf{M}$ cannot be uniquely determined: Let us consider matrices $\mathbf{A}_1, \mathbf{B}_2$ where

$$\mathbf{A}_1 = \begin{bmatrix} 3 & 3 \\ 8 & 7 \end{bmatrix}, \mathbf{B}_1 = \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}$$

and

$$\mathbf{M} = \mathbf{A}\mathbf{B}^T = \mathbf{A}_1{\mathbf{B_1}}^T$$

This is a manifestation of the **rotational freedom** associated with bilinear models that is, one may rotate $\mathbf{A}$ by right-multiplying with any non-singular matrix $\mathbf{T}$ and counter-rotate $\mathbf{B}^T$ (left-multiplying with $\mathbf{T^{-1}}$) without modifying $\mathbf{X}$. In the example above,

$$\mathbf{T} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

and the inverse is

$$\mathbf{T}^{-1} = \begin{bmatrix} -\frac{1}{3} & \frac{2}{3} \\ \frac{2}{3} & -\frac{1}{3} \end{bmatrix}$$

One can make sure that this rotational property holds for the above example even when the factors are positive. It is also remarkable that, even though T^{-1} has negative elements, the resulting product turns out to be non-negative. Thus, even under non-negativity constraints, the bilinear decomposition is not unique in general.

3.4.2 NMF Identifiability Properties

In the previous section, we demonstrated that, even when non-negativity holds, the bilinear model is not unique in general. However, in [20] we can find a study that explores the conditions under which a Non-negative Matrix Factorization may be unique. In [20], a geometric representation of NMF is provided. According to that representation, all elements of $\mathbf{M}$ are mapped to a simplicial cone defined by the columns of $\mathbf{B}$. In conclusion, [20] states three conditions and an example, under which the Non-negative Matrix Factorization can be uniquely determined. Further discussion about this topic lies beyond the scope of the present text.

Chapter 4

Sparse Matrix Regression

4.1 Introduction to Linear Regression

Linear Regression aims to model variables using linear combinations of certain observations or measurements. The observations are usually called *predictors* and are symbolized by $\mathbf{x}$ and the outputs are called *responses* and are symbolized by y. Linear Regression provides an estimate $\hat{y}$ of the actual output. The notation for this introductory section will be the one used in the related bibliography. However, when we reach the point to describe the process of Matrix Factorization via Sparse Regression, we will provide the required transformation among notations.

Consider the input vector

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix}$$

The linear regression model has the form:

$$\hat{y} = f(\mathbf{x}) = \beta_0 + \sum_{j=1}^{p} x_j \beta_j \tag{4.1}$$

We assume that the relationships between $\mathbf{x}$ and y are linear or approximately linear. The parameters $\{\beta_i\}$ are initially unknown and are computed via a training process. Let us denote as $\mathbf{X}$ the $N \times p$ input matrix, with each row containing an input vector. Let us denote as $\mathbf{y}$ the vector with the corresponding outputs to each row of $\mathbf{X}$. The most popular estimator for $\{\beta_i\}$ is the least squares estimator which minimizes the *residual sum of squares* (RSS):

$$RSS(\beta) = \sum_{i=1}^{N} (\mathbf{y}_i - f(\mathbf{x}_i))^2 = \sum_{i=1}^{N} (\mathbf{y}_i - \sum_{j=1}^{p} \mathbf{X}_{ij}\beta_j))^2$$

We denote as $\mathbf{X}$ the $N \times (p+1)$ training matrix, where each row corresponds to a data vector and $\mathbf{y}$ as the $N \times 1$ vector of outputs. We can rewrite RSS as:

$$RSS(\beta) = \| \mathbf{y} - \mathbf{X}\beta \|^2 = (\mathbf{y} - \mathbf{X}\beta)^T (\mathbf{y} - \mathbf{X}\beta)$$

$$\frac{\partial RSS}{\partial \beta} = -\mathbf{2}\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) \Rightarrow$$

$$\frac{\partial^2 RSS}{\partial \beta^2} = \mathbf{2}\mathbf{X}^T\mathbf{X}$$

We assume that, $\mathbf{X}$ being full column rank, $\mathbf{X}^T\mathbf{X} > 0$ and $(\mathbf{X}^T\mathbf{X})^{-1}$ exists, thus by setting $\frac{\partial RSS}{\partial \beta} = 0$ we obtain the unique solution:

$$\beta_{LS} = (\mathbf{X}^T\mathbf{X})^{-\mathbf{1}}\mathbf{X}^T\mathbf{y} \tag{4.2}$$

4.2 Shrinkage Least Squares Methods

Instead of just minimizing the Residual Sum of Squares function, the group of shrinkage least squares methods also introduces an additional penalty to the cost function which eventually leads to more interpretable models by shrinking or discarding some of the predictor values. In the following subsections we shall review two widely accredited shrinkage methods, the **Ridge regression** and the **Lasso regression**.

4.2.1 Lasso Regression

The Lasso is a shrinkage and selection method for linear regression and was first introduced in [4]. The lasso minimizes the RSS, penalizing the optimization process with the absolute value of the coefficients. By posing this constraint, some coefficient tend to become 0, thus providing a more conceptually interpretable model.

The setting for lasso is the same as the one described in the linear regression model. We assume a $N \times p$ matrix $\mathbf{X}$ containing the predictor values and a vector $\mathbf{y}$ containing the responses. We also denote

$$\hat{\beta} = \begin{bmatrix} \hat{\beta}_{\mathbf{1}} \\ \vdots \\ \hat{\beta}_p \end{bmatrix}$$

Then, the lasso estimate $(\hat{a}, \hat{\beta})$ is obtained through the following optimization formula:

$$\min_{\beta}\{\sum_{i=1}^{N}(\mathbf{y}_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{i,j})^2\} \tag{4.3}$$
$$\text{subject to } \sum_{j=1}^{p}|\beta_j| \leq t$$

The upper bound t is a tuning parameter. When t is large enough, the constraint has no effect and the solution is just the usual multiple linear least squares regression of $\mathbf{y}$ on $\mathbf{X}$. However when for smaller values of t the solutions are shrunken versions of the least squares estimates, justifying the term "shrinkage methods". Often, some of the coefficients β_j are zero, a property that is quite often desirable. Choosing t is like choosing the number of predictors to use in a regression model. In Chapter 6, we offer methods for choosing this parameter, with respect to our application.

An alternative way to express the lasso equation is:

$$\min_{\beta}\{\sum_{i=1}^{N}(\mathbf{y}_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{i,j})^2 + \lambda\sum_{j=1}^{p}|\beta_j|\} \tag{4.4}$$

with $\lambda \geq 0$ playing an equivalent role to t in equation 4.3.

4.2.2 Ridge Regression

Ridge Regression, being a shrinkage least square regression method, additionally to minimizing the RSS, poses a penalty on the size of the regression coefficients. The additional constraint comes with the form of the squared value of the coefficients.

Let us assume a $N \times p$ matrix $\mathbf{X}$ containing the predictor values and a vector $\mathbf{y}$ containing the responses. Then, the Ridge regression estimate is obtained by solving the following optimization formula:

$$\min_{\beta}\{\sum_{i=1}^{N}(\mathbf{y}_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{i,j})^2 + \lambda\sum_{j=1}^{p}\beta_j^2\} \tag{4.5}$$

Equivalently, we may solve:

$$\min_{\beta}\{\sum_{i=1}^{N}(\mathbf{y}_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{i,j})^2\} \tag{4.6}$$
$$\text{subject to } \sum_{j=1}^{p}\beta_j^2 \leq t$$

4.2.3 Comparison of Ridge & Lasso Regression methods

As we stated earlier, in the Lasso regression case, it occurs more often, some of the regression coefficients to be exactly zero. This quality is desirable in our application as it will be made clear in the next chapters. At this point, we will provide insight as to why Lasso instead of Ridge regression produces coefficients equal to zero.

The penalty for each of the two regression methods comes with the flavor of an additive factor of a p-norm in the optimization formula. In the case of Ridge regression, we have the squares of the coefficients, hence the norm that is added is $p = 2$. However, in the case of Lasso regression, the absolute values of the coefficients are added as penalty, hence the norm in that case is $p = 1$. The insight that we shall provide is based upon the way that each type of norm behaves. We notice that in equations 4.3 and 4.5 (or 4.4 and 4.6 in respect), the only thing that changes is the penalty. This means that the difference in the regression coefficients produced, should be caused by the difference in the penalty.

The criterion

$$\sum_{i=1}^{N}(\mathbf{y}_i - \sum_{j}\beta_j x_{i,j}) \tag{4.7}$$

equals the quadratic function

$$(\beta - \hat{\beta})\mathbf{X}^T\mathbf{X}(\beta - \hat{\beta}) \tag{4.8}$$

The above quadratic function corresponds to elliptical contours which are centered at the least squares estimates $\hat{\beta}$. Moreover, for each of the two regression methods, we have to examine how the penalty is mapped in space. One way to do this is by employing the depiction of a p-norm to the L^P space. It can be shown that the $l-1$ norm of a vector is being depicted as a unary rectangle, where the $l-2$ norm can be depicted as a unary circle. The illustration for the $l-1$ norm type of norm can be shown in the following figure:

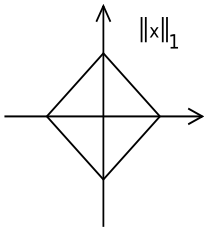
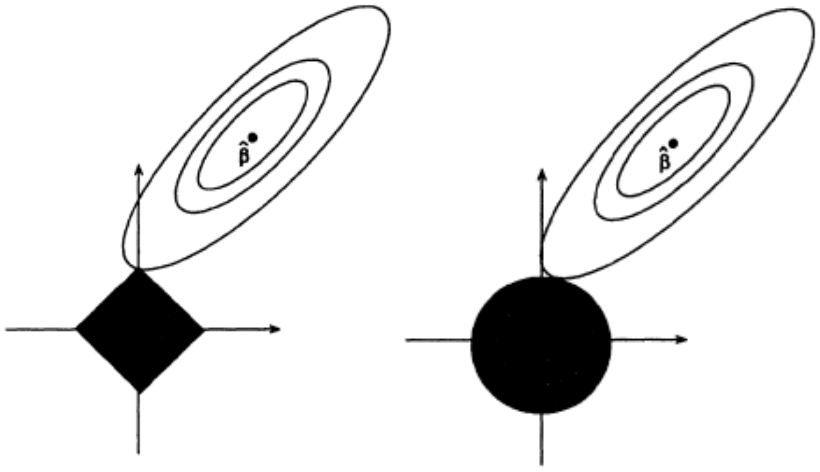


Figure 4.1: The $l-1$ norm in the L^P space. (Copyright belongs to Wikipedia.org)

When we plot the penalty and the quadratic function that was shown above, we observe in the figure below that in the case of Lasso regression, when the contours of the quadratic intersect the $l-1$ norm rectangle, this occurs at a corner, producing a zero coefficient. On the other hand, in the Ridge regression case, the contours intersect the unary circle in points that produce coefficients close to but not exactly zero. This consists no proof but provides good insight for this property of Lasso regression. A proof for that argument lies beyond the scope and the purpose of the present thesis.



(a) Lasso Regression (b) Ridge Regression

Figure 4.2: Lasso and Ridge comparison in space. (Copyright belongs to [4])

4.3 Matrix Factorization with Sparse Regression

The goal of Matrix Factorization using Sparse Regression is to factor the data matrix $\mathbf{M}$ in two sparse factors:

$$\mathbf{M} \approx \mathbf{A}\mathbf{B}^T \tag{4.9}$$

A matrix is called sparse if the zero elements largely outnumber the non zero elements.

We examined the case of Non-negative Matrix Factorization that poses non-negativity constraints to factors $\mathbf{A}, \mathbf{B}$. In Sparse Matrix Regression, our main objective is to produce factors that maintain a fundamental characteristic of the initial matrix, **sparsity**. In certain applications, this property is useful as it reflects prior knowledge about the "true" factors. E.g. Reviewer profiles are by construction sparse.

The method chosen to accomplish this objective is Lasso regression, which was previously described in detail. This method apart from solving the factorization problem that we encountered on a previous chapter, adds an additional constraint to the minimization formula. This constraint comes with the form of a $l-1$ norm penalty added to the objective function of the minimization process.

In it's simplest form, the Sparse Matrix Regression problem can be stated as an Lasso regression problem as follows:

$$\min_{\mathbf{B}}\{\| \mathbf{M} - \mathbf{A}\mathbf{B}^T \|_F^2 + \lambda \| \mathbf{B} \|_1\} \tag{4.10}$$

The above equation is similar to the equation introduced by Lasso Regression, if we consider each column of $\mathbf{M}$ as an output vector, matrix $\mathbf{A}$ as the input matrix and each column of $\mathbf{B}$ as the β_j coefficients we need to estimate. The latter provides us with the appropriate notation transformation.

Additionally,we may use matrix transposition to reform our regression formula in order to penalize $\| \mathbf{A} \|_1$ and pose sparsity in $\mathbf{A}$

$$\min_{\mathbf{A}}\{\| \mathbf{M}^T - \mathbf{B}\mathbf{A}^T \|_F^2 + \lambda \| \mathbf{A} \|_1\} \tag{4.11}$$

Both equations 4.10 and 4.11 assume a fixed value for $\mathbf{A}$ and $\mathbf{B}$ respectively. However, in many applications (such as the present), there is the need to force sparsity constraints to both factors of the data matrix. This need rises because both factors are conceptually supposed to be sparse thus posing sparsity constraints to just one of them may lead to unwanted results, such as total zeroing of the "sparse" matrix in order to conform to the constraint. This can be expressed by the following optimization formula:

$$\min_{\mathbf{A},\mathbf{B}}\{\| \mathbf{M} - \mathbf{A}\mathbf{B}^T \|_F^2 + \lambda \| \mathbf{A} \|_1 + \lambda \| \mathbf{B} \|_1\} \tag{4.12}$$

One may additionally want to pose non-negativity constraints to $\mathbf{A}$ and $\mathbf{B}$ (e.g for better intuitive interpretation). In that case, 4.12 can be restated as:

$$\min_{\mathbf{A},\mathbf{B}}\{\| \mathbf{M} - \mathbf{A}\mathbf{B}^T \|_F^2 + \lambda \| \mathbf{A} \|_1 + \lambda \| \mathbf{B} \|_1\} \tag{4.13}$$
$$\text{subject to } a_{i,j} \geq 0 \text{ and } b_{i,j} \geq 0$$

In both $l-1$ penalties of equation 4.13, the parameter λ has to be the same, otherwise, the algorithm might be led to instabilities.

The problem 4.12 is not linear thus it cannot be solved in this direct form. One way to solve this is by solving two linear lasso problems in an alternating fashion (in a similar way to ALS-NMF and TALS for PARAFAC). The complexity per iteration for Algorithm 4.1 is $\mathcal{O}(IJ\hat{k}^2)$. The Alternating Sparse Regression algorithm pseudocode follows:

Algorithm 4.1 Alternating Sparse Matrix Regression

Input: $\mathbf{M}$ of size $I \times J$, $\hat{k}$,λ
Output: $\mathbf{A}, \mathbf{B}$
  $\mathbf{A}$=randn(I,$\hat{k}$);
  $\mathbf{B}$=randn(J,$\hat{k}$);
  while convergence criterion is not met **do**
    $\mathbf{B}=\underset{\mathbf{B}}{\text{argmin}}\{\| \mathbf{M} - \mathbf{A}\mathbf{B}^T \|_F^2 + \lambda \| \mathbf{B} \|_1\}$
    $\mathbf{A}=\underset{\mathbf{A}}{\text{argmin}}\{\| \mathbf{M}^T - \mathbf{B}\mathbf{A}^T \|_F^2 + \lambda \| \mathbf{A} \|_1\}$
  end while

Chapter 5

Tensors and Tensor Decompositions

5.1 An Introduction to Tensors

A tensor or N-way array is a multidimensional array. To define the term *way* consider the following example. A vector is also called one-way array and a matrix is also called a two-way array. Consequently, a three-way array is a three dimensional structure that is composed by a number of matrices (or two-way arrays). In this thesis, we will be concerned with three-way arrays, though higher order arrays or tensors can be defined and all of the following apply to them. Before proceeding, we must note that tensors or n-way arrays in the present context should not be confused with tensors in the field of quantum mechanics or physics in general. From now on when referring to a tensor we refer to a three way array, a structure indexed by 3 indices (i.e $I \times J \times K$).

The applications of tensors and tensor decomposition span from Signal Processing to Data Mining, from Computer Vision to Chemometrics, Psychometrics and others. A very informative bibliographic list of tensor applications can be found in [5]. Additionally, most of the material regarding PARAFAC comes from [1] (chapter 6).

5.2 Definitions

Definition 10 (Kronecker Product)**.** *Given two matrices* $\mathbf{A}, \mathbf{B}$ *with sizes* $I \times F_1$ *and* $J \times F_2$ *in respect, we define the* $IJ \times F_1F_2$ *matrix as* $\mathbf{A} \otimes \mathbf{B}$ *the Kronecker product:*

$$\mathbf{A} \otimes \mathbf{B} := \begin{bmatrix} \mathbf{B}a_{1,1} & \cdots & \mathbf{B}a_{1,F_1} \\ \vdots & \ddots & \vdots \\ \mathbf{B}a_{I,1} & \cdots & \mathbf{B}a_{I,F_1} \end{bmatrix}$$

Example 3. *Consider matrices*

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 5 & 10 \\ 15 & 20 \\ 25 & 30 \end{bmatrix}$$

The Kronecker product is

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} 5 & 10 & 10 & 20 \\ 15 & 20 & 30 & 40 \\ 25 & 30 & 50 & 60 \\ 15 & 30 & 20 & 40 \\ 45 & 60 & 60 & 80 \\ 75 & 90 & 100 & 120 \end{bmatrix}$$

□

Definition 11 (Khatri Rao product)**.** *Given two matrices* $\mathbf{A}(I \times F_1)$ *and* $\mathbf{A}(J \times F_1$ *with equal number of columns, we define the Khatri-Rao product as*

$$\mathbf{A} * \mathbf{B} = \left[\mathbf{a_1} \otimes \mathbf{b_1} \cdots \mathbf{a}_F \otimes \mathbf{b}_F\right]$$

where $\mathbf{a}_f$ *and* $\mathbf{b}_f$ *denote the* f*-th column of* $\mathbf{A}$ *and* $\mathbf{B}$*.*

Example 4. *Consider matrices*

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 5 & 10 \\ 15 & 20 \\ 25 & 30 \end{bmatrix}$$

The Khatri-Rao product is

$$\mathbf{A} * \mathbf{B} = \begin{bmatrix} 5 & 20 \\ 15 & 40 \\ 25 & 60 \\ 15 & 40 \\ 45 & 80 \\ 75 & 120 \end{bmatrix}$$

□

5.3 The PARAFAC decomposition

5.3.1 Introduction

Let us reconsider the definition of matrix rank, following the bilinear decomposition that was presented in a previous chapter 7. That was one way to define the matrix rank. It's value is found in the fact that this same definition can be used in order to define a three way array rank.

Let us now consider an $I \times J \times K$ three-way array $\mathcal{X}$ with typical element $x_{i,j,k}$, and the F -component trilinear decomposition:

$$x_{i,j,k} = \sum_{f=1}^{F} a_{i,f} b_{j,f} c_{k,f} \tag{5.1}$$

for all $i = 1 \cdots I$, $j = 1 \cdots J$, and $k = 1 \cdots K$. The above equation expresses the three-way array $\mathcal{X}$ as a sum of F rank-one three-way factors. This decomposition can be visualized in figure 5.3.1. Analogous to the definition of matrix rank, we conclude to the following definition:

Definition 12 (Three-way array rank)**.** *A three-way array* $\mathcal{X}$ *rank can be defined as the minimum number of rank-one (three-way) components needed to decompose* $\mathcal{X}$*.*

An example of a rank one three way array can be found in figure 5.3.1 below.

Depending on context, the vectors $\mathbf{a}_f, \mathbf{b}_f, \mathbf{c}_f$ are often referred to as loading vectors or score vectors or factor profiles in the multi-way literature.

5.4 PARAFAC Theory

5.4.1 Notation and Preliminaries

In this section we define some notation that is deemed necessary, in order to proceed with the introduction of the PARAFAC model. In the following lines we will review methods of visualization of a three way array as a two way array (matrix) and preliminary information regarding PARAFAC.

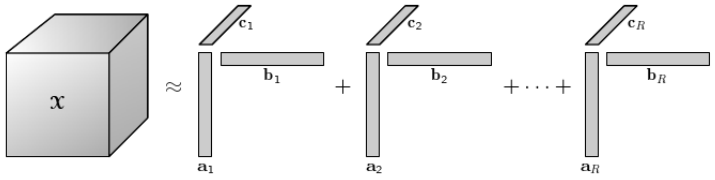


Figure 5.1: The trilinear decomposition of a three way array. (Copyright belongs to [5])

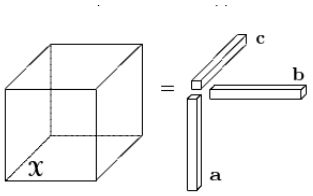


Figure 5.2: A rank-one three way array. (Copyright belongs to [5])

Le us define an $I \times F$ matrix $\mathbf{A}$ with typical element $\mathbf{A}(if) := a_{i,f}$, $J \times F$ matrix $\mathbf{B}$ with typical element $\mathbf{B}(jf) := b_{j,f}$, and $K \times F$ matrix $\mathbf{C}$ with typical element $\mathbf{C}(kf) := c_{k,f}$. Furthermore, define $J \times K$ matrices $\mathcal{X}_i$, $I \times K$ matrices $\mathcal{X}_j$, and $I \times J$ matrices $\mathcal{X}_k$ with corresponding typical elements $\mathcal{X}_i(jk) := \mathcal{X}_j(ik) := \mathcal{X}_k(ij) := x_{i,j,k}$. Then the model in Equation 5.1 can be written in three different ways in terms of systems of simultaneous matrix equations (each of which can be interpreted as "slicing" the three-way array $\mathcal{X}$ along one of the three dimensions:

$$\mathcal{X}_i = \mathbf{B}\mathbf{D}_i(\mathbf{A})\mathbf{C}^T, i = 1 \cdots I \tag{5.2}$$

$$\mathcal{X}_j = \mathbf{A}\mathbf{D}_j(\mathbf{B})\mathbf{C}^T, j = 1 \cdots J \tag{5.3}$$

$$\mathcal{X}_k = \mathbf{A}\mathbf{D}_k(\mathbf{C})\mathbf{B}^T, k = 1 \cdots K \tag{5.4}$$

where $\mathbf{D}_i(\mathbf{A})$ denotes a diagonal matrix constructed out of the i-th row of $\mathbf{A}$. $\mathbf{D}_j(\mathbf{B})$ and $\mathbf{D}_k(\mathbf{C})$ are constructed accordingly.

The PARAFAC model can be said to be the analogous of the Singular Value Decomposition (SVD) to the realm of three way arrays. The one property that makes PARAFAC and the trilinear decomposition differ significantly from the bilinear decomposition is the fact that the trilinear decomposition is proved to be unique under some mild conditions that are going to be discussed further later on. Another alternative to PARAFAC in terms of generalization of the SVD for higher than matrix order arrays is the **Tucker model** or **HO-SVD**, abbreviation of High Order Singular Value Decomposition, which was first proposed in 1966 in [21] . This model will not concern us in terms of this thesis, but a reference in this point was essential for completeness. The Tucker model can be expressed as:

$$\mathcal{X}_{I \times J \times K} = [\mathbf{A}_{I \times R}, \mathcal{G}_{R \times S \times T}, \mathbf{C}_{K \times T}, \mathbf{B}_{J \times S}] \tag{5.5}$$

In 5.5, the subscripts indicate the dimensions of each component. Matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ are generally orthonormal and full column rank. Instead of PARAFAC, the Tucker model decomposes a tensor into a set of matrices and a smaller core tensor. As we will see later on, the Tucker model can be useful for the computation of the PARAFAC model (5.4.3).

As it will be shown in the following example, the PARAFAC model does not suffer from rotational freedom that was exhibited in 3.4. In order to see how PARAFAC excludes subspace rotation, let us reconsider the example shown on the Matrix Factorization chapter:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 2 & 3 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 3 & 3 \\ 4 & 5 \end{bmatrix}$$
$$\mathbf{X} = \mathbf{A}\mathbf{B}^{\mathbf{T}} = \begin{bmatrix} 6 & 9 \\ 15 & 23 \end{bmatrix}$$

We can easily see that given $\mathbf{A}, \mathbf{B}$, $\mathbf{X}$ cannot be uniquely determined: Let us consider matrices $\mathbf{A}_1, \mathbf{B}_2$ where

$$\mathbf{A}_1 = \begin{bmatrix} 3 & 3 \\ 8 & 7 \end{bmatrix}, \mathbf{B}_1 = \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}$$

and

$$\mathbf{X} = \mathbf{A}\mathbf{B}^T = \mathbf{A}_1{\mathbf{B}_\mathbf{1}}^T$$

This is a manifestation of the **rotational freedom** associated with bilinear models that is, one may rotate $\mathbf{A}$ by right-multiplying with any non-singular matrix $\mathbf{T}$ and counter-rotate $\mathbf{B}^T$ (left-multiplying with $\mathbf{T}^{-\mathbf{1}}$) without modifying $\mathbf{X}$. In the example above,

$$\mathbf{T} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

Now let us examine a similar example, but instead of a matrix and it's bilinear decomposition, we use three-way array and the PARAFAC decomposition.

Example 5. *Consider the* $2 \times 2 \times 2$ *three-way array with the following two slices:*

$$\mathcal{X}_1 = \begin{bmatrix} 6 & 9 \\ 15 & 23 \end{bmatrix}, \mathcal{X}_2 = \begin{bmatrix} 6 & 0 \\ 0 & 7 \end{bmatrix}$$

$$\mathbf{B}^{\mathbf{T}} = \begin{bmatrix} 39 & 59 \\ 99 & 153 \end{bmatrix}$$

If we define

$$\mathbf{C} = \begin{bmatrix} 1 & 1 \\ 6 & 7 \end{bmatrix}$$

then we can express the two slices as

$$\mathcal{X}_1 = \mathbf{A}\mathbf{D}_\mathbf{1}(\mathbf{C})\mathbf{B}^T$$
$$\mathcal{X}_2 = \mathbf{A}\mathbf{D}_\mathbf{2}(\mathbf{C})\mathbf{B}^T$$

□

The above example shows that it is possible for the trilinear decomposition to achieve uniqueness in terms of subspace rotation.

In addition to the slice representations in 5.2,5.3,5.4, there exist several more ways of representing a three-way array $\mathcal{X}$ into a matrix. For example, we have two more kinds of representations, a block-row and a block-column matrix form. The block-row representation is

$$\mathcal{X}^{(I \times JK)} = \begin{bmatrix} \mathcal{X}_{k=1} & \cdots & \mathcal{X}_{k=K} \end{bmatrix} \tag{5.6}$$

and the block-column representation is

$$\mathcal{X}^{(JI \times K)} = \begin{bmatrix} \mathcal{X}_{i=1} \\ \vdots \\ \mathcal{X}_{i=I} \end{bmatrix} \tag{5.7}$$

The superscript in each case denotes at first the size of the resulting matrix. Moreover, in each representation, it denotes which index "runs" faster along the rows or columns. For example, in $\mathcal{X}^{(I\times JK)}$, the j-th index runs faster than the k-th index along the rows (as J goes first in the product JK) and in $\mathcal{X}^{(JI\times K)}$, the j-th index (that goes first in the product JI) runs faster than the i-th index along the columns.

5.4.2 Identifiability and Uniqueness

As mentioned before, a very important property of the PARAFAC decomposition is uniqueness. This uniqueness property is proved to hold under mild conditions. This means that factor matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ don't suffer from matrix ambiguities such as the ones discussed previously regarding the bilinear decomposition.

[22, 23]

5.4.3 Algorithms for Fitting the PARAFAC model

Alternating Least Squares

The principle of Alternating Least Squares (ALS) regression, previously introduced in 3.3.2 can be applied to the computation of the trilinear decomposition. The basic idea behind ALS is simple: In each step

- update one of the three matrices, using least squares regression based on previous estimates
- update the other matrices
- repeat from the top until convergence

The ALS principle for the trilinear decomposition is generally called TALS (trilinear ALS). The trilinear ALS method is appealing primarily because it is guaranteed to converge monotonically, but also because it is conceptually simple (no parameters to tune, each step solves a standard LS problem) and provides good performance.

The optimization formula and and the update formula for the algorithm can be derived by using the block-column matrix representation 5.7 as follows:

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} \| \begin{bmatrix} \hat{\mathcal{X}}_{i=1} \\ \vdots \\ \hat{\mathcal{X}}_{i=I} \end{bmatrix} - \begin{bmatrix} \mathbf{B}\mathbf{D_1}(\mathbf{A}) \\ \vdots \\ \mathbf{B}\mathbf{D}_I(\mathbf{A}) \end{bmatrix} \mathbf{C}^T \|_F^2 \tag{5.8}$$

The conditional least squares update for $\mathbf{C}$ is

$$\hat{\mathbf{C}^{\mathbf{T}}} = \begin{bmatrix} \hat{\mathbf{B}}\mathbf{D_1}(\hat{\mathbf{A}}) \\ \vdots \\ \hat{\mathbf{B}}\mathbf{D}_I(\hat{\mathbf{A}}) \end{bmatrix}^{\dagger} \begin{bmatrix} \hat{\mathcal{X}}_{i=1} \\ \vdots \\ \hat{\mathcal{X}}_{i=I} \end{bmatrix} \tag{5.9}$$

where $\hat{\mathbf{A}}, \hat{\mathbf{B}}$ denote the estimated values of $\mathbf{A}, \mathbf{B}$. The LS updates for $\mathbf{A}, \mathbf{B}$ can be obtained the same way, considering the symmetry of the trilinear model.

For efficiency, some TALS algorithms in the literature employ line search/relaxation to speed up convergence. The complexity per iteration for TALS is $\mathcal{O}(IJKF)$.

Compression/COMFAC

COMFAC stands for COMplex parallel FACtor analysis and utilizes compression of the initial three-way aray, in order to speed up the computation of the PARAFAC model. COMFAC involves the 3 following steps:

1. Compression of the three-way array
2. Initialization and fitting of the PARAFAC model in the compressed array
3. Decompression and refining of the solution in the original space

5.5 Other Tensor Decompositions

When describing the equivalence of PARAFAC to the matrix SVD, we stumbled upon an alternative model, the Tucker model. Some alternative models, other than the PARAFAC and the Tucker model are:Tucker2, PARAFAC2 ([24]), CANDELINC ([25]), DEDICOM, PARATUCK2 ([26]) and INDSCAL ([27]).

5.6 Tensor Applications

There is a vast number of Tensor applications, ranging from Chemometrics to Psychometrics and from Signal Processing to Data Mining. In the present section we will present two widely known and cited tensor applications, coming from the field of Data Mining, which is the area of interest for the present thesis. The copyright of every image taken from a particular publication belongs to the respective owner and is reproduced here providing a reference to the original source.

5.6.1 TOPHITS

The TOPHITS (abbreviation for Topical-HITS) is considered to be a generalization of the HITS link analysis algorithm described in chapter 3, in the Three-way realm. The TOPHITS algorithm was first proposed in [6] and eventually every image regarding this algorithm is taken from that particular publication.

Opposed to the matrix based HITS algorithm (3.2.3) tophits creates a three way array as the Adjacency Structure (the term Adjacency Matrix is not appropriate in the present context), that also incorporates relations between terms contained in the web pages. This information is represented by the third dimension of the tensor.

The TOPHITS algorithm defines the following $I \times J \times K$ tensor $\mathcal{X}$:

$$x_{i,j,k} = \begin{cases} 1 & \text{if page i points to page j using term k} \\ 0 & \text{otherwise} \end{cases} \qquad (5.10)$$

$$\text{where } 1 \leq i, j \leq I, 1 \leq k \leq K$$

5.6.2 Discussion Tracking

An example of Discussion Tracking using Tensors and the PARAFAC decomposition can be found in [7]. The particular publication uses the e-mail database of the Enron company, that closed down on December 2, 2001 in order to extract and detect discussions over a period of time from e-mail messages.

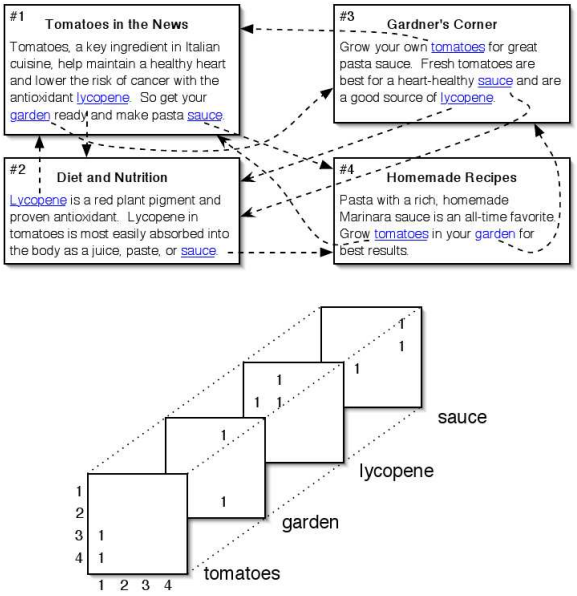


Figure 5.3: Example of a TOPHITS Adjacency Tensor (Copyright belongs to [6])

This publication models the email data as a three way adjacency array of dimension $I \times J \times K$ where I is the number of terms parsed from the messages, J is the number of e-mail authors and K is the period of time. The authors constructed two different arrays, one for a daily period of time, resulting to $K = 357$ and another one for a monthly period of time, resulting to $K = 12$. These arrays were decomposed using the PARAFAC decomposition as described earlier in the present chapter and the PARAFAC decomposition with Non-negativity constraints (also found as Non-negative Tensor Factorization (NNTF) in the literature).

The following images provide a timetable produced by the PARAFAC factors that describes the extracted conversation topics over time. The images were taken from [7].

5.7 Software

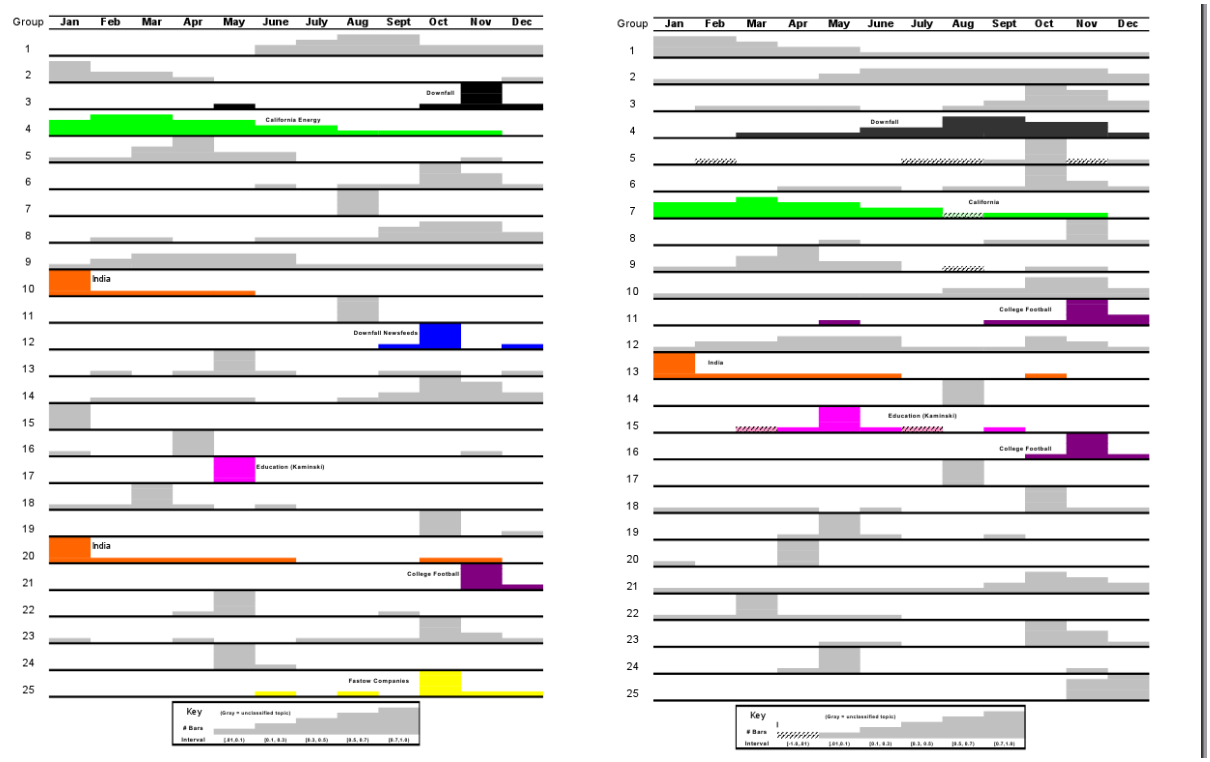
5.7.1 Tensor Toolbox for Matlab

The Tensor Toolbox for Matlab is a toolbox developed by Sandia National Laboratories. It provides interface for the manipulation of dense or sparse tensors, using the object oriented principle that was newly introduced to Matlab. Apart from the ease of data manipulation, the Tensor Toolbox provides among others, implementation of the PARAFAC model and the Tucker model and variations of both.

The Tensor Toolbox for Matlab can be downloaded from [28]. Additional publications regarding the toolbox can be found in [29, 30].

5.7.2 The N-way Toolbox for Matlab

The N-way Toolbox for Matlab [31] is a freely available collection of functions and algorithms for modeling multiway data sets by a range of multilin- ear models. Apart from the PARAFAC model,more types of multiway models are covered: multilinear partial least-squares regression (PLSR), generalized



(a) NNTF (b) PARAFAC

Figure 5.4: Topics extracted over time from the Enron Email dataset. (Copyright belongs to [7])

rank annihilation method (GRAM), direct trilinear decomposition (DTLD) and the class of Tucker models.

Regarding the computation of the PARAFAC model that is of interest in this particular thesis, the N-way Toolbox for Matlab embeds some optional constraints into the least-squares error minimization algorithm such as orthogonality, unimodality and non-negativity constraints.

The N-way Toolbox for Matlab can be downloaded from `http://www.models.life.ku.dk/source/nwaytoolbox/`.

Chapter 6

Proposed Algorithms

In this chapter, we present the work done on this thesis. The flow of the presentation is according to the actual flow of the algorithms developed. First, we describe the task of data acquisition and assembly, in order to form our working dataset. In that section, we also introduce a web mining tool we developed especially for the purposes of this thesis. Later on, we present the formulation of the data in algebraic form and finally, we provide the developed algorithms and associated details.

6.1 GoogleScholar Miner

GoogleScholar miner is a small Java based application, we developed in order to query Google Scholar [32] for a specific researcher, browse through the resulting list of publications, download it and store it in a database, in a form that can be easily processed and made use of. Google Scholar [32] is service offered by Google that provides a researcher with the ability to search online for a publication of a broad area of scientific fields.

The idea for GoogleScholarMiner was inspired by a citation analysis application called *Publish or Perish* ([33]). Publish or Perish, like GoogleScholarMiner, uses the search results of Google Scholar in order to perform citation analysis for a researcher and obtain his/her publication list. However, we chose to design our own miner because Publish or Perish cannot be used in a non-manual way and the results it produces, regarding the publication list, were very noisy, usually containing a lot of information irrelevant to the actual title of the publication). The application we designed for the purposes of this thesis proved to produce more clean results.

6.1.1 Method of Retrieval

The format of the Google Scholar results is plain machine generated HTML, making it a bit frustrating to recover successfully, without nuisance (in the form of irrelevant text) the desired information about each reviewer. However, for each publication entry of the results, apart from the usual information (number of citations, external links etc.) contains a link to BL-Direct (abbreviation for British Library Direct). This link, is an entry of that publication to the online repositories of BL-Direct. Moreover, this entry contains all the information useful to us, namely the title of the publication and the year of publication. Apart from the title (which is essential for obvious reasons), the number of citations and the year of publication prove to be useful to determining the weight of each publication of a researcher, in terms of "field of expertise". A more detailed discussion regarding the field of expertise and the publication weighting scheme will be presented later on. The number of citations can be extracted from the original results page. The basic idea behind the retrieval process of a list of publications, consists of the following steps:

1. Provide Google Scholar with a query containing the full name of the researcher and preferable scientific field, in order to disambiguate-as much as possible-the results. It is also more efficient

to insert just the first letter of the first name and the whole last name of the author, in order to disambiguate references to the same person using variations of his/her first name (e.g Peter/Pete)

2. Browse systematically through the whole list of publications.

3. For each publication entry, retrieve the number of citations. Then follow the link to the BL-Direct page and retrieve the title and year of publication.

6.1.2 The Algorithm

In pseudocode form, the GoogleScholarMiner can be stated as follows:

Algorithm 6.1 GoogleScholarMiner algorithm

Input: The name of a researcher his/her scientific field.
Output: The list of publications and respective date and citation number for that researcher.

```
1: Make a query to Google Scholar using the name and the scientific field.
2: for all result pages do
3:    Browse through the page and search for BL-Direct links
4:    for all BL-Direct links do
5:       Follow the BL-Direct link.
6:       Parse the resulting page, storing the title and the year of publication.
7:       Store the citation number that belongs to the same structure of code corresponding to that link
8:    end for
9: end for
```

6.2 Data Acquisition

In this section, we shall describe the way of acquiring the data needed for the completion of our task. The first step is to extract a very large number of raw information regarding these two entities: the papers and the reviewers. The list of submitted papers is already available to us from the TPC chair. So is the list of the prospective reviewers. For the case of the reviewers, the challenge is to extract raw terms starting only with their first and last name and obtaining somehow a list of their publication history.

The terms for each entity were extracted using either a custom approach (which puts to work most of the preliminaries described on chapter 2) or the KEA tool (that was briefly described on that same chapter). We found out that the raw and final extracted terms produced using KEA were slightly more accurate, though we present both methods here, for completeness.

6.2.1 Retrieving Terms from the Submitted Papers

As we mentioned before, the list of submitted papers is already available to us from the conference organizers. Thus, there is no need to look for supplementary data for this process. We remind the reader that the information we need to extract, relies solely on the papers' titles. Throughout the following lines, we describe the custom method used and how KEA was used, in order to extract these raw initial terms:

Custom Method

The custom method used consists of the following steps:

1. Parsing & Preprocessing: This is the process of parsing the paper titles, breaking every sentence to seperate words (tokenizing) and eliminating stopwords using a custom made list that is oriented to the broad specific scientific area of the conference.

2. Creation of the paper-by-term matrix, based on the terms that "survived" the stopword elimination. The weighting scheme employed is the TF×IDX scheme. Note that the frequency of occurence of every term to a paper's title can be at most 1 almost surely: The event of finding twice a term (that is not a stopword) on a single title is nearly impossible. The paper-by-term matrix is created using only single word terms.

3. Extraction of single word terms using the paper-by-term matrix. We can additionally use the SVD approximation of the paper-by-term matrix in a lower dimension, in order to benefit from all the qualities of Latent Semantic Indexing.

4. Extraction of multi word terms: This process requies at first an additional step of text pre-processing : **Part-of-speech tagging** (POS tagging). POS tagging is procedure of marking each token of the text with the appropriate part of speech tag (e.g verb, noun, adjective e.t.c). This step is essential for the application the C-value/NC-value methods. For POS tagging, we use the POS tagger developed by the Stanford University. More information regarding this tool (and a download link) can be found in [34]. When done with the POS tagging process, we proceed to the extraction of multi word terms, using the NC-value method which was described on chapter 2.

The resulting final list of raw terms is the union of the single-word terms set and the multi-word term set obtained from the above steps.

KEA

When using the KEA tool, things are not as complicated as above. What needs to be done is provide the tool with a set of training documents, in order to train the Naive Bayes model and then feed the application with the paper titles, in order to extract the desired raw terms. For training, we used the abstract of published papers in the same broad area as the conference (e.g. Signal Processing), using as author defined keyphrases the index terms of each paper. The model built upon this training data is the one used throughout this whole thesis.

The next step, is the extraction process. We choose to concatenate all the titles in one big document, as it is stated on the KEA documentation that the algorithm works better on long input text files. The length of the extracted terms is set to at most two words. This, of course, is a design parameter and can be altered at will, with respect to the scientific field of interest (e.g. medical terms often consist of more than two consequent words).

6.2.2 Retrieving Terms from the Reviewers

The case of the Reviewers is identical to the case of the Papers, with a twist. The list of publications for each reviewer is not availabe as is the list of the submitted papers. The list of publications for each reviewer, along with the number of citations for each publication and the date of each publication are the means that lead us to the extraction of the initial terms for the prospective reviewers. As this information is not readily accessible, we need to employ GoogleScholarMiner. For each reviewer, we use GoogleScholarMiner in order to access the Google Scholar database, inquire the list publications of that reviewer and finally process the results of that inquiry to gain access to that reviewer's term profile.

Let us examine the case for one reviewer, which can be generalized for the complete list of reviewers. The output of GoogleScholarMiner for that specific reviewer is the list of publications accompanied with the associated publication date and the number of citations for each publication. The last two pieces of information are essential in order to discover his area of expertise and his current research interests. The publications that correspond to the area of expertise and are of his current research interest should be the most likely to produce representative terms. Taking this into account, we devised a weighting mechanism that "boosts" publications that belong either to the area of expertise or to the current research interests or both, in order to bias the raw extraction procedure to pick more terms from these publications.

This mechanism is simple and intuitive: We take into account the timeliness of the publication and the impact it has had over it's respective scientific field. We initially start with equal weights for all publications. If a publication is very recent (e.g. published in the last 5 years), it is more likely that this publication comes under the current research interests of that reviewer, thus we grant the publication's weight with an additive factor of 4. Moreover, if a publication is highly cited, this probably means that this publication was very influential and comes under the field of expertise of that reviewer. In that case, we grant the corresponding weight with an additive factor of 5. Given all of the above, if a publication is timely, it's weight is 4, if it is highly cited, it's weight is 5 and if it exhibits both properties, it's weight is 10. An example of this weight vector is illustrated on the following figure:

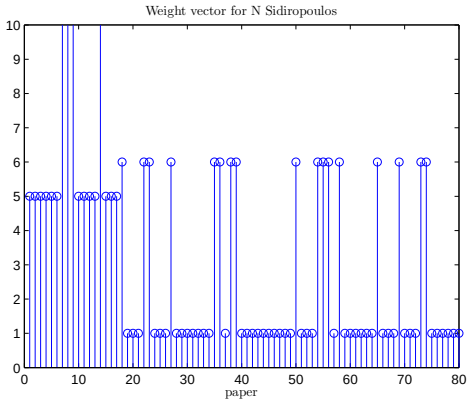


Figure 6.1: Weight vector for Nicholaos Sidiropoulos

Apart from the application of the weight vector of each reviewer, the whole extraction process, either following the custom method or using KEA, is essentially the same as the one involving the submitted papers. The weight vector is applied on each case as follows:

- Custom Method: The paper-by-term matrix represents all the papers from all reviewers' publications lists in the Vector Space Model representation. For convenience, let us assume that the lists of publications are vertically concatenated, forming the list of all publications among the reviewers. For each set of n papers (corresponding to the list of publications of the reviewer p), we also have a weight vector $\mathbf{w}_p$ with size $n \times 1$. We have R such vectors (with variable size, according to the size of each reviewer's publication list), where R is the total number of reviewers. Let us denote as $\mathbf{w}$ the vector produced by the vertical concatenation of all $\mathbf{w}_p, p = 1 \cdots R$. Then, the final weighted paper-by-term matrix is given by:
$$\hat{\mathbf{M}} = \text{diag}(\mathbf{w})\mathbf{M}$$
where diag($\mathbf{w}$) denotes a diagonal matrix with diagonal elements equal to the given vector $\mathbf{w}$, $\mathbf{M}$ denotes the original paper-by-term matrix for all the reviewers and $\hat{\mathbf{M}}$ denotes the weighted one.

- KEA: When using KEA, we don't have access to the weighting scheme of the algorithm applied. What we can control is the input text. This gives us the opportunity to bias the outcome of the extraction process by repetition. This means that on the input text that contains all the paper titles for a reviewer, we repeat each title as many times as the corresponding paper's weight.

6.2.3 How many raw terms should we extract?

The question of this section's title poses the fundamental trade off that emerges from this process. The number of raw terms that we extract for both entites is a parameter that is reflected later on the results of the final extracted terms. If the number of raw terms is significantly small, then each profile is

very sparse and some of the algorithms proposed later on may produce unwanted results. On the other hand, if the number of raw terms is vast, the quality of the resulting list of terms might be compromised by noisy terms that come about as a result of the large number of initial raw terms. This trade off is also discussed on the final chapter where we attempt to find the "sweet spot" between the number of initial raw terms and the quality of the final results.

6.3 Data Formulation

In this section we take a moment to describe the notation be used throughout all three algorithms that follow. These are:

- '*reviewer_terms*': This is the list of raw terms retrieved using the GoogleScholarMiner application. The number of these terms is a tunable parameter and it's value is obviously reflected on the cardinality of the final results.
- '*paper_terms*': This is the list of raw terms retrieved from the titles of the submitted papers. The number of these terms is a tunable parameter and it's value is also reflected on the final results, as in the case of '*reviewer_terms*'.
- '*all_terms*' This is the union of the 2 above sets and constitutes our set of index terms.
- **P**: This matrix is the $P \times T$ paper-by-term matrix that uses a basis '*all_terms*'. P denotes the number of submitted papers and T denotes the number of terms in '*all_terms*', throughout this thesis.
- **R** : This matrix is the $R \times T$ reviewer-by-term matrix that uses a basis '*all_terms*'. R denotes the number of the reviewers.

We also need to define the terms **concept/topic**: As a topic or concept we define a group of terms that have similar meaning and constitute a conceptual category and exists in the latent term space.

6.4 Profiling using Non-negative Matrix Factorization

6.4.1 Why NMF?

When introducing Matrix Factorization, we reviewed two different ways of tracking the factorization of the data matrix **M** in the form

$$\hat{\mathbf{M}} \approx \mathbf{A}\mathbf{B}^T$$

These two different methods were the SVD of **M** or the Non-negative Matrix Factorization of **M**. Even though the approximation of a matrix in a lower dimension is optimal when using the truncated SVD of that matrix (in the least squares sense), this property does not suffice to model our data. This happens, because optimality in the least squares sense is not our main objective. Our main desire is model interpretability, a property that Matrix Factorization using the SVD lacks. The SVD does not provide any guarantees of non-negativity on the singular vectors. This is not desirable in the present applications for the following reasons:

1. We need to express each reviewer's profile as a linear combination (or sum-of-parts) of the rows of **B**. A sum of parts concept is not valid when negative parts exist.
2. There is good chance that in the resulting profile matrix there exist negative elements, a property that does not make intuitive sense in the present context. If a reviewer is not matched at all by a specific term, the weight of that term for that reviewer should be zero. Any value less than zero has no logical interpretation. Despite that fact, the original data matrix **M** is binary, thus non-negative and it is highly desirable for the resulting matrix to maintain the properties of the initial data matrix.

As a result of the above, the choice that comes naturally is NMF. The model produced by NMF exhibits non-negativity as the original data matrix does ,and provides interpretability. Even though the NMF is not an optimal approximation (non-negativity constraints lead to non-convexity, thus the result most probably comes from a local minimum), the benefit of the interpretability outweighs the non-optimality in terms of minimization and in the least squares sense.

6.4.2 The Algorithm

At first, using the initial data matrices $\mathbf{R}, \mathbf{P}$, we create the data matrix $\mathbf{M}$ such that

$$\mathbf{M} = \begin{bmatrix} \mathbf{R} \\ \mathbf{P} \end{bmatrix}$$

which denotes the vertical concatenation of $\mathbf{R}, \mathbf{P}$. In this step we unify the data that represent reviewers and papers, as we aim to extract terms that best represent both entities. It is valid to concatenate $\mathbf{R}, \mathbf{P}$ vertically, as they share the same basis vector.

After creating $\mathbf{M}$, the next step is to obtain it's Non-negative Matrix Factorization, the two factors $\mathbf{A}, \mathbf{B}$. The number of bilinear components for the factorization (or equivalently the approximating low rank of the target matrix) is denoted by $\hat{k}$. This parameter is tunable; a reasonable value for $\hat{k}$ is the total number of concepts/topics we want to extract.

The NMF of $\mathbf{M}$ is $\{\mathbf{A}, \mathbf{B}\}$ where $\mathbf{A} \in \{\mathbb{R}^{R+P\times\hat{k}}, a_{i,j} \geq 0\}$ and $\mathbf{B} \in \{\mathbb{R}^{T\times\hat{k}}, b_{i,j} \geq 0\}$. The columns of $\mathbf{B}$ are the basis vectors of the $\hat{k}$-dimensional latent term space. This means that each one of these vectors, represents an entire group of terms. The rows of $\mathbf{A}$ are the profiles of each entity in that $\hat{k}$-dimensional space.

From the above factorization, we can reconstruct each profile as a linear combination (sum-of-parts) of the columns of $\mathbf{B}$ with the profile vector in the latent term space. The reconstructed profile matrix for the reviewers is

$$\hat{\mathbf{R}} = \mathbf{A}(1 : R, :)\mathbf{B}^T \tag{6.1}$$

where $\mathbf{A}(1 : R, :)$ denote the first R rows of $\mathbf{A}$. Respectively, the new profiles matrix for the papers is

$$\hat{\mathbf{P}} = \mathbf{A}(R + 1 : R + P, :)\mathbf{B}^T \tag{6.2}$$

where $\mathbf{A}(R + 1 : R + P, :)$ denotes the rows of $\mathbf{A}$ from $R + 1$ to $R + P$. This can be also written as $\mathbf{A}(R + 1 : end, :)$ in Matlab notation. It should be obvious that $\text{rank}(\hat{\mathbf{R}}) \leq \hat{k}$ and $\text{rank}(\hat{\mathbf{P}}) \leq \hat{k}$, if $\hat{k} \leq R$ and $\hat{k} \leq P$.

The new profile matrices $\hat{\mathbf{R}}, \hat{\mathbf{P}}$ contain the scores of each reviewer or paper in a lower dimension. Let us recall at this point the Latent Semantic Indexing principle which states that such a depiction can deal with synonymy and polyesmy issues. Moreover, such a depiction can cluster together in a way reviewers or papers which have a similar profile in the basis 'all_terms'.

Let's consider at this point the case of $\hat{\mathbf{R}}$. Each row of that matrix is a vector representing the corresponding reviewer in the $\hat{k}$-dimensional space. When plotting such a vector, we may observe peaks with varying values that represent high scoring terms and points with smaller values that can be seen either as noise or as less significant terms. Additionally, all of the elements of that vector are non-negative. In order to get the highest scoring terms, we may first determine the highest peaks of the vector. In other words, **post-processing** is needed, in the form of sorting each profile vector.

After sorting each profile vector, we take the $\hat{t}$ highest values and we collect the corresponding terms from '*all_terms*'. This procedure is done repeatedly for all the reviewers and the resulting terms for each reviewer are stored along with the others'. When this process is finished, we have a set of terms selected to represent best each reviewer. The parameter $\hat{t}$, just like $\hat{k}$ is a tunable parameter which essentially determines the size of the resulting set of terms. If we choose for example $\hat{t} = 15$ or accordingly small, the resulting list of terms should be significantly smaller than the initial raw list of '*all_terms*'. Let us call the resulting list of terms, obtained the way we described earlier as '$final\_reviewer\_terms$'.

Now, all we need is to filter out all those terms that don't represent the papers and keep only those that do. One way to do this is by taking the set intersection of '$final\_reviewer\_terms$' and the '*paper_terms*'. In that way, we discard all the terms in '$final\_reviewer\_terms$' that don't appear on any title and keep only the ones that jointly describe both the reviewers and the papers. This method is universal among the three algorithms described in this chapter. This final set of terms is denoted as $\underline{\mathbf{T}}$.

After determining the final list of terms $\underline{\mathbf{T}}$, we have to compute the matrices $\hat{\mathbf{R}}$ and $\hat{\mathbf{P}}$. These matrices are the reviewer-by-term and paper-by-term matrices produced using as a basis the terms in $\underline{\mathbf{T}}$. The concept behind the computation is very simple: we just take the columns of $\mathbf{R}$ and $\mathbf{P}$ that correspond to the terms in $\underline{\mathbf{T}}$. But there is a twist. The initial feature matrices $\mathbf{R}, \mathbf{P}$ do not represent any latent relationships between reviewers and terms or papers and terms, hence it doesn't come as a surprise that some of the rows of $\hat{\mathbf{R}}$ and $\hat{\mathbf{P}}$ contain only zeros. This property is inevitable but fortunately, the algorithm itself provides the solution. The basic idea is that for every zero row (equivalently, every empty profile), we find the most similar reviewer or paper and copy that profile to the empty one.

Let us consider the case of $\hat{\mathbf{R}}$ and then the case of $\hat{\mathbf{P}}$ is completely analogous. Suppose that there exist in $\hat{\mathbf{R}}$ r rows that contain all-zero elements. Now consider the matrix

$$\mathbf{S}_r = \hat{\mathbf{R}}\hat{\mathbf{R}}^T \tag{6.3}$$

This matrix is of size $R \times R$ and is defined as the correlation matrix of the reviewer profiles.The element $s_{r_{i,j}}$ shows the similarity of reviewer i with the reviewer j. The matrix $\mathbf{S}_r$ is obviously, by definition symmetric. For each of the r rows of $\hat{\mathbf{R}}$ that contain zeros, we find, using the similarity matrix $\mathbf{S}_r$, the most similar reviewer with non-empty profile and copy that profile to the empty one. The similarities expressed by $\mathbf{S}_r$ take into account relationships between reviewers, as the matrix used to compute $\mathbf{S}_r$ is the one obtained by the factorization.The case of $\hat{\mathbf{P}}$ is completely analogous, with the similarity matrix being defined as

$$\mathbf{S}_p = \hat{\mathbf{P}}\hat{\mathbf{P}}^T \tag{6.4}$$

In the Algorithm 6.2 table we provide a summary of the proposed algorithm in a concise, pseudocode form.

The algorithm used to track the Non-negative Matrix Factorization equation is the Multiplicative Method algorithm that can be found on the according chapter (3.1).

6.5 Profiling using Sparse Matrix Regression

6.5.1 Why Sparse Matrix Regression?

As we reviewed earlier, on the Data Acquisition section, the number of "batch" terms initially extracted from the web and the submitted papers is vast. This fact tends to produce very sparse profile vectors, because a reviewer is depicted in the large dimensional space by very few non-zero coordinates (representing his profile). Consequently, the data matrix $\mathbf{M}$ tends to be very sparse. From this point on, a matrix or vector is said to be **sparse** when the zero elements strongly outnumber the non-zero elements. In this thesis, we don't employ an explicit way of calculating the sparsity of a matrix or a vector, other than the ratio of the non-zero elements over the zero elements. Even though there exist

Algorithm 6.2 Profile Extraction using Non-negative Matrix Factorization

Input: List of submitted papers and reviewers
Output: $\underline{\mathbf{T}}, \tilde{\mathbf{R}}, \tilde{\mathbf{P}}$

paper_terms=extract raw terms for submitted papers
reviewer_terms=$\emptyset$
for all reviewers in the reviewer list **do**
  reviewer_terms=*reviewer_terms* $\cup$ Terms{GoogleScholarMiner(reviewer)}(see algorithm 6.1)
end for
all_terms=*paper_terms* $\cup$ *reviewer_terms*
construct $\mathbf{P}, \mathbf{R}$
set $\mathbf{M} = \begin{bmatrix} \mathbf{R} \\ \mathbf{P} \end{bmatrix}$
$\{\mathbf{A}, \mathbf{B}\}$=Non-negative Matrix Factorization ($\mathbf{M}$,$\hat{k}$)(see Algorithm 3.1)
$\hat{\mathbf{R}}$=$\mathbf{A}(1:R,:)\mathbf{B}^T$
$\hat{\mathbf{P}}$=$\mathbf{A}(R+1:end,:)\mathbf{B}^T$
terms=$\emptyset$
for $i = 1 : R$ **do**
  $\hat{\mathbf{r}}_i$ =$\hat{\mathbf{R}}(i,:)$
  Sort $\hat{\mathbf{r}}_i$ and get the $\hat{t}$ highest scoring terms.
  Then, get the corresponding terms and store them in 'w'
  terms=*terms* $\cup$ w
end for
$\underline{\mathbf{T}}$=*terms* $\cap$ *paper_terms*
Construct $\tilde{\mathbf{R}} \in [0,1]^{(R \times \hat{k})}$ as
$\tilde{\mathbf{R}}$=$\mathbf{R}$(:,indices of '*terms*' in vector '*all_terms*')
if $\tilde{\mathbf{R}}$ contains zero rows **then**
  Construct similarity matrix $\mathbf{S_r}$=$\hat{\mathbf{R}}\hat{\mathbf{R}}^T$
  Set diagonal elements of $\mathbf{S_r}$ equal to 0.
  for each zero row of $\tilde{\mathbf{R}}$ **do**
    Get the reviewer most similar (with non-empty profile) to the one with
    the empty profile and copy the non empty profile
    to the empty one.
  end for
end if
Construct $\tilde{\mathbf{P}} \in [0,1]^{(P \times \hat{k})}$ as
$\tilde{\mathbf{P}}$=$\mathbf{P}$(:,indices of '*terms*' in vector '*all_terms*')
if $\tilde{\mathbf{P}}$ contains zero rows **then**
  Construct similarity matrix $\mathbf{S_p}$=$\hat{\mathbf{P}}\hat{\mathbf{P}}^T$
  Set diagonal elements of $\mathbf{S_p}$ equal to 0.
  for each zero row of $\tilde{\mathbf{P}}$ **do**
    Get the paper most similar (with non-empty profile) to the one with
    the empty profile and copy the non zero profile
    to the empty one.
  end for
end if

alternative ways of measuring sparsity, the above definition suffices for the characterization of our data matrix as sparse.

When decomposing the data matrix $\mathbf{M}$ into factors $\mathbf{A}, \mathbf{B}$, using the bilinear decomposition and more specifically Non-negative Matrix Factorization as shown on the previous section, there are no guarantees whatsoever that the resulting profile matrix will maintain the sparsity the original matrix $\mathbf{M}$ exhibits. In fact, for Non-negative Matrix Factorization, we have observed that nearly no element is exactly zero. By looking at the profile matrix resulting from NMF, we can see the peaks of each profile vector that represent the highest scoring terms for each reviewer and the rest of the elements being of lower-but not zero-value. This leads to weak interpretability of the resulting matrix as it is. This means that for the NMF profile matrix, in order to obtain the most representative terms for each reviewer, post-processing is needed, as shown on the previous section.

In Sparse Matrix Regression, we propose an alternative Matrix Factorization that uses Sparse Regression as shown in chapter 5, in order to produce more interpretable profiles. By employing Sparse Matrix Regression, we make sure that the resulting matrix will be sufficiently sparse, inheriting this quality from the original matrix, and we eliminate the need of post-processing, as the algorithm itself does the "filtering" of the noise that the plain matrix factorization produces. In the following section we will describe in detail the algorithmic developments made for this method.

6.5.2 The Algorithm

The algorithm we propose in table 6.3 is similar on the basic elements of algorithm 6.2 but differs fundamentally on the fact that uses Sparse Matrix Regression (SMR) in order to enforce sparsity on the resulting factors.

As we reviewed earlier, the data matrix $\mathbf{M}$ is factored by $\{\mathbf{A}, \mathbf{B}\}$ where $\mathbf{A} \in \{\mathbb{R}^{R+P\times\hat{k}}, a_{i,j} \geq 0\}$ and $\mathbf{B} \in \{\mathbb{R}^{T\times\hat{k}}, b_{i,j} \geq 0\}$; additionally, both factors are forced to be sparse. The columns of $\mathbf{B}$ are the basis vectors of the $\hat{k}$-dimensional latent term space. This means that each one of these vectors, represents an entire group of terms. Due to sparsity, each column should have very few nonzero elements, leading to less noisy compound term vectors, which finally lead to less noisy reconstructed profiles for each reviewer and paper. Therefore, there is no need at all for post-processing of the results (sorting each profile vector and picking the $\hat{t}$ highest scoring terms),

Note that in contrast to the NMF case, where we pick exactly $\hat{t}$ high scoring terms for each reviewer, in SMR the number of candidate terms varies from reviewer to reviewer, a property that makes tremendous intuitive sense. Take for example a reviewer that has a very large number of publications (which should lead to an equally large profile) and a reviewer that has a significantly lower number of publications. When we choose to pick an equal number of terms from both profiles, we enter ourselves into a trade-off. If we set the number of terms to pick to a very large number, we might be extracting a lot of noise from the reviewer with the small profile and accordingly, if we set that number to a small value, we might be excluding very important terms from the reviewer large profile.

The algorithm used to compute the Sparse Matrix Regression is the alternating algorithm shown in Algorithm 4.1. The algorithm used to solve each one of the sub problems of the alternating sparse matrix regression is an element-wise coordinate descent. This algorithm can be found in pseudocode in Algorithm 6.4.

When presenting the NMF based algorithm, we encountered the problem of an empty profile to the final reviewer-by-term matrix or the final paper-by-term matrix. This problem was solved by computing similarities of the empty profiles to non-empty profiles, using the matrices produced directly by $\{\mathbf{A}, \mathbf{B}\}$. If a row of the matrix produced by $\{\mathbf{A}, \mathbf{B}\}$ is zero, this row cannot provide us with any useful information

regarding the similarity of that profile to other profiles. The point made from this statement is that there is a difference between empty profiles in the matrix obtained by the decomposition directly and empty profiles emerging in the final matrices $\tilde{\mathbf{P}}, \tilde{\mathbf{R}}$. An empty profile in $\tilde{\mathbf{P}}, \tilde{\mathbf{R}}$ can be dealt with information directly from $\{\mathbf{A}, \mathbf{B}\}$, where an empty profile in $\hat{\mathbf{M}} = \mathbf{A}\mathbf{B}^T$ is highly undesirable.

As we reviewed earlier, empty profiles resulting in matrices $\tilde{\mathbf{P}}, \tilde{\mathbf{R}}$ can be dealt with information directly from $\{\mathbf{A}, \mathbf{B}\}$. However, in the SMR case, if the value of λ is very high, sparsity on $\hat{\mathbf{M}} = \mathbf{A}\mathbf{B}^T$ may cause some rows of $\hat{\mathbf{M}}$ to be completely zero. This case resembles the one described above and can also be dealt with by obtaining similarities and substituting empty profiles accordingly. However, an empty reconstructed profile is slightly more undesirable from an empty profile in matrices $\tilde{\mathbf{P}}, \tilde{\mathbf{R}}$. Empty profiles in $\hat{\mathbf{M}}$ will most likely result to less terms that match both reviewers and papers. Therefore, it is desirable but not definitive that none or very few reconstructed profiles are empty.

This issue can be dealt with an appropriate choice of the parameters λ and $\hat{k}$, parameters that control the sparsity of the reconstructed profiles. A formal way to choose among reasonable combinations of those parameters is a Monte Carlo simulation, as exhibited in Algorithm 6.5. In that pseudocode, $\underline{\mathcal{K}}$ denotes a set of candidate values for $\hat{k}$ and $\underline{\Lambda}$ denotes a set of candidate values for λ. Finally, $z_{\hat{k},\lambda}$ denotes the number of zero rows in the resulting matrix for a specific couple of $\{\hat{k}, \lambda\}$ and $n_{\hat{k},\lambda}$ denotes the average number of nonzero elements per row of the resulting matrix for a specific couple of $\{\hat{k}, \lambda\}$. Usually the decision for $\{\lambda, \hat{k}\}$ is taken by empirical evaluation, perhaps after obtaining some possible choices using simulation.

Algorithm 6.3 Profile Extraction using Sparse Matrix Regression

Input: List of submitted papers and reviewers
Output: $\underline{\mathbf{T}}, \tilde{\mathbf{R}}, \tilde{\mathbf{P}}$

1: *paper_terms*=extract raw terms for submitted papers
2: *reviewer_terms*=$\emptyset$
3: **for all** reviewers in the reviewer list **do**
4: *reviewer_terms*=*reviewer_terms* $\cup$ Terms{GoogleScholarMiner(reviewer)}(see algorithm 6.1)
5: **end for**
6: *all_terms*=*paper_terms* $\cup$ *reviewer_terms*
7: construct $\mathbf{P}, \mathbf{R}$
8: set $\mathbf{M} = \begin{bmatrix} \mathbf{R} \\ \mathbf{P} \end{bmatrix}$
9: Choose $\{\hat{k}, \lambda\}$(see Algorithm 6.5)
10: $\{\mathbf{A}, \mathbf{B}\}$=Sparse Matrix Regression ($\mathbf{M}$,$\hat{k}$,λ)(see Algorithms 6.4,4.1)
11: $\hat{\mathbf{R}}$=$\mathbf{A}(1:R,:)\mathbf{B}^T$
12: $\hat{\mathbf{P}}$=$\mathbf{A}(R+1:end,:)\mathbf{B}^T$
13: *terms*=$\emptyset$
14: **for** $i = 1:R$ **do**
15: $\hat{\mathbf{r}}_i$ =$\hat{\mathbf{R}}(i,:)$
16: Get the corresponding terms of $\hat{\mathbf{r}}_i$
17: (e.g all non-zero positions of the vector) and store them in 'w'
18: terms=terms $\cup$ w
19: **end for**
20: $\underline{\mathbf{T}}$=*terms* $\cap$ *paper_terms*
21: Construct $\tilde{\mathbf{R}} \in [0,1]^{(R \times \hat{k})}$ as
22: $\tilde{\mathbf{R}}$=$\mathbf{R}$(:,indices of '*terms*' in vector '*all_terms*')
23: **if** $\tilde{\mathbf{R}}$ contains zero rows **then**
24: Construct similarity matrix $\mathbf{S_r}$=$\hat{\mathbf{R}}\hat{\mathbf{R}}^T$
25: Set diagonal elements of $\mathbf{S_r}$ equal to 0.
26: **for** each zero row of $\tilde{\mathbf{R}}$ **do**
27: Get the reviewer most similar (with non-empty profile) to the one with
28: the empty profile and copy the non empty profile
29: to the empty one.
30: **end for**
31: **end if**
32: Construct $\tilde{\mathbf{P}} \in [0,1]^{(P \times \hat{k})}$ as
33: $\tilde{\mathbf{P}}$=$\mathbf{P}$(:,indices of '*terms*' in vector '*all_terms*')
34: **if** $\tilde{\mathbf{P}}$ contains zero rows **then**
35: Construct similarity matrix $\mathbf{S_p}$=$\hat{\mathbf{P}}\hat{\mathbf{P}}^T$
36: Set diagonal elements of $\mathbf{S_p}$ equal to 0.
37: **for** each zero row of $\tilde{\mathbf{P}}$ **do**
38: Get the paper most similar (with non-empty profile) to the one with
39: the empty profile and copy the non zero profile
40: to the empty one.
41: **end for**
42: **end if**

Algorithm 6.4 Element-wise Coordinate Descent for Non-negative Sparse Matrix Regression

while convergence criterion is not met **do**
  for $j = 1 : J$ **do**
    for $f = 1 : F$ **do**
      $\mathbf{d} = \mathbf{M}(:, j) - \mathbf{AB}(j, :)^T + \mathbf{A}(:, f)\mathbf{B}(j, f)$
      $\mathbf{a} = \mathbf{A}(:, f)$
      if $(\mathbf{a^T d} - \frac{\lambda}{2}) > 0$ **then**
        $\mathbf{B}(j, f) = \frac{\mathbf{a}^T\mathbf{d} - \frac{\lambda}{2}}{\|\mathbf{a}\|_2^2}$
      else
        $\mathbf{B}(j, f) = 0$ {Non-negativity constraint}
      end if
    end for
  end for
end while

Algorithm 6.5 Monte Carlo simulation for the selection of λ and $\hat{k}$ for Sparse Matrix Regression

for $\hat{k} \in \underline{\mathcal{K}}$ **do**
  for $\lambda \in \underline{\Lambda}$ **do**
    $\{\mathbf{A}, \mathbf{B}\}$=Alternating Non-negative SMR($\mathbf{M}, \hat{k}, \lambda$)
    $\hat{\mathbf{M}} = \mathbf{AB}^T$
    Count how many zero rows $\hat{\mathbf{M}}$ has and store it in $z_{\hat{k},\lambda}$.
    Count the average number of non-zero elements per row of $\hat{\mathbf{M}}$ and store it in $n_{\hat{k},\lambda}$.
  end for
end for
Choose according to $z_{\hat{k},\lambda}$ and $n_{\hat{k},\lambda}$ the best $\{\hat{k}, \lambda\}$ that maintains sparsity and does not produce empty profiles if possible.

6.6 Profiling using PARAFAC

6.6.1 Why PARAFAC?

The main reason to model our data as a three way array instead of a matrix is, as we reviewed earlier, on the chapter discussing the PARAFAC decomposition, the PARAFAC model does not generally suffer from rotational freedom and indeterminacy, in contrast to the bilinear decomposition and the Matrix Factorization technique.

Moreover, the representation of the data as a three way array provides an alternative intuitive way of modeling the Reviewer and Paper profiling problem. The third dimension of the terms connecting a reviewer with a submitted paper is what enabled us to express this problem as a Three-way array.

In the literature there are a couple of recent examples of three way array modeling of data, in data and text mining applications. Some of these examples can be found in:[35, 36, 6, 37]. We presented two applications in the previous chapter. However, no publication so far has approached the Reviewer-Paper profile extraction problem as a three-way analysis application.

6.6.2 The Algorithm

The three way array is constructed using $\mathbf{P}$ as the initial value of each of the P slices of the array. Every slice of the three-way array is essentially the Reviewer-Term matrix with emphasis given on the keywords that appear on the corresponding paper. More specifically, let us consider the i-th reviewer and the j-th term. If the i-th reviewer is matched by that term (equivalently $\mathbf{R}(i, j) = 1$) and that term

appears on the corresponding paper, then the value of that element in that slice is 1. If $\mathbf{R}(i,j) = 1$ but the term does not appear on the paper then the (i,j) element of the slice is 0.25

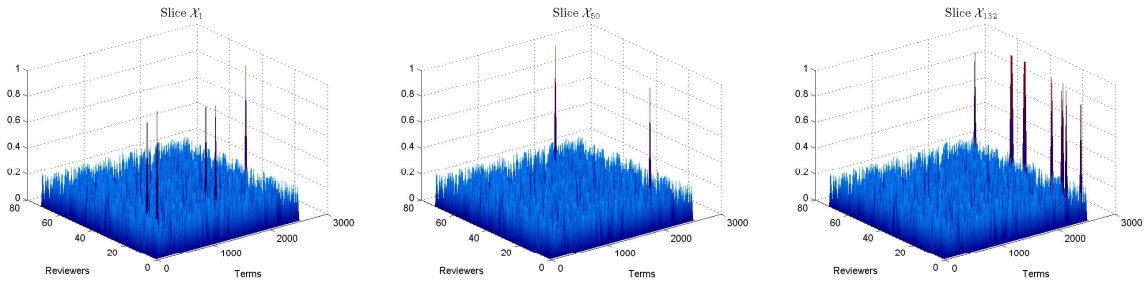
A typical definition of $\mathcal{X}$ is :

$$\mathcal{X}(:,:,p) = \mathbf{R} \odot \mathbf{Q}_p, p = 1 \dots P \tag{6.5}$$

where

$$\mathbf{Q}_p = \begin{cases} 1 & \text{if } \mathbf{P}(p,k) = 1 \\ \frac{1}{4} & \text{else} \end{cases} \tag{6.6}$$

Following the method described above, the three-way array $\mathcal{X}$ is constructed. We can see an example of the slices of $\mathcal{X}$ in figure 6.6.2 below:



(a) Slice 1 (b) Slice 50 (c) Slice 132

Figure 6.2: Slices of $\mathcal{X}$

After forming $\mathcal{X}$, we have to obtain the trilinear or PARAFAC decomposition of this three way array. There is no need to determine the rank of $\mathcal{X}$. We will decompose $\mathcal{X}$ to a small number of factors instead. This number of factors is the dimension of the latent term space and roughly corresponds to the total number of concepts or topics that we need to extract. A way of determining the number of factors for the decomposition can be found on Algorithm 6.6 and resembles the one used in Sparse Matrix Regression. Again, this simulation is not definitive in picking an appropriate value for $\hat{k}$; it only provides a guideline as to which values of $\hat{k}$ behave in a desirable manner.

Let us denote the number of factors for the PARAFAC decomposition as $\hat{k}$. With the decomposition of $\mathbf{X}$ to $\hat{k}$ factors using PARAFAC, we obtain $\hat{k}$ triplets (rank one three way arrays) that each one represent the score or loading values for each of the concepts that we will extract. The PARAFAC decompositions produces matrices $[\mathbf{A}, \mathbf{B}, \mathbf{C}]$ with sizes $R \times \hat{k}$, $T \times \hat{k}$ and $P \times \hat{k}$ respectively. Matrix $\mathbf{A}$ contains the scores of each Reviewer for every latent term. Matrix $\mathbf{B}$ contains the score of each normal term in the latent term space and finally, matrix $\mathbf{C}$ contains the scores of each paper for every latent term. To be precise, matrix $\mathbf{C}$ cannot provide the same amount of useful information as matrices $\mathbf{A}, \mathbf{B}$, as the third mode of $\mathcal{X}$ is derived from a Hadamard product, that produces differences in scaling in different slices of $\mathcal{X}$.

Consequently, we shall use matrices $\mathbf{A}, \mathbf{B}$ to reconstruct our profile matrix. As we mentioned earlier, each row of matrix $\mathbf{A}$ contains the scores for each reviewer in the $\hat{k}$-dimensional term space, whereas each row of $\mathbf{B}$ contains the scores for the corresponding term in the $\hat{k}$-dimensional term space. This leads us to the reconstruction of the profile vector for each reviewer. If we denote the profile vector for the i-th reviewer as $\hat{\mathbf{r}}_i^T$, its value is obtained by taking the linear combination of all the rows of $\mathbf{B}$(or equivalently the columns of $\mathbf{B^T}$) weighed by the elements of the i-th row of $\mathbf{A}$. This can be written in the following equation as

$$\hat{\mathbf{r}}_i^T = \mathbf{A}(i,:)\mathbf{B}^T, i = 1 \cdots R \tag{6.7}$$

where, $\mathbf{A}(i,:)$ denotes the i-th row of $\mathbf{A}$.

Later on, we may organize all the profile vectors in a $R \times T$ matrix $\mathbf{R_p}$, where the i-th row is the profile vector of the i-th reviewer. Equivalently, we have

$$\hat{\mathbf{R}}(i,:) = \mathbf{p}_i^T = \mathbf{A}(i,:)\mathbf{B}^T, i = 1 \cdots R \tag{6.8}$$

$$\hat{\mathbf{R}} = \mathbf{A}\mathbf{B}^T \tag{6.9}$$

Algorithm 6.6 Monte Carlo simulation for the selection of $\hat{k}$ for PARAFAC

for $\hat{k} \in \underline{\mathcal{K}}$ **do**
  $\{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$=PARAFAC($\mathcal{X}, \hat{k}$,non-negativity)
  $\hat{\mathbf{M}} = \mathbf{A}\mathbf{B}^T$
  Count how many zero rows $\hat{\mathbf{M}}$ has and store it in $z_{\hat{k}}$.
  Count the average number of non-zero elements per row of $\hat{\mathbf{M}}$ and store it in $n_{\hat{k}}$.
end for
Choose according to $z_{\hat{k}}$ and $n_{\hat{k}}$ the best $\hat{k}$ that maintains sparsity if possible and does not produce empty profiles if possible.

In order to compute the PARAFAC decomposition, we use the N-way Toolbox for Matlab ([31]). The algorithm the toolbox uses to track the decomposition is the TALS algorithm. We additionally pose non-negativity constraints to all three modes of the decomposition. This means that all three factor matrices $[\mathbf{A}, \mathbf{B}, \mathbf{C}]$ are non-negative. As exhibited on the NMF proposed algorithm, non-negativity is essential for the proper interpretation of the model.

The way to obtain the final keywords is the same as the one described on the two previous algorithms: we take the intersection of the high scoring terms for each reviewer, with the terms that belong to paper titles.

A significant property that we observed is that the results produced tend to be sparse in a satisfactory manner. In contrast to Sparse Matrix Regression, the PARAFAC approach does not provide any control over sparsity, but usually the resulting matrix resembles more to the SMR one rather than the NMF one. This implies that post-processing is usually not necessary, since sparsity tends to single out the most important terms. However, we cannot exclude the fact of a case where post-processing can be useful to produce more clear results. In conclusion, post-processing of the results is optional and should be done only if really needed, a decision that can usually be made by inspection and empirical evaluation of the results.

Another difference from the other two proposed algorithms is the fact that the similarity matrix $\mathbf{S_p}$ cannot be constructed using $\hat{\mathbf{P}}$ as there is no way of constructing this matrix. Instead, we use the original matrix $\mathbf{P}$ and the paper similarity matrix is

$$\mathbf{S_p} = \mathbf{P}\mathbf{P}^T \tag{6.10}$$

Algorithm 6.7 Profile Extraction using PARAFAC with Non-negativity Constraints

Input: List of submitted papers and reviewers
Output: $\underline{\mathbf{T}}, \tilde{\mathbf{R}}, \tilde{\mathbf{P}}$

paper_terms=extract raw terms for submitted papers
reviewer_terms=$\emptyset$
for all reviewers in the reviewer list **do**
  reviewer_terms=*reviewer_terms* $\cup$ Terms{GoogleScholarMiner(reviewer)}(see algorithm 6.1)
end for
all_terms=*paper_terms* $\cup$ *reviewer_terms*
construct $\mathbf{P}, \mathbf{R}$
construct $\mathcal{X}$:
for $p = 1 : T$ **do**
  for $i = 1 : T$ **do**
    if $\mathbf{P}(p, k) == 1$ **then**
      $\mathbf{Q}_p = 1$
    else
      $\mathbf{Q}_p = \frac{1}{4}$
    end if
  end for
  $\mathcal{X}(:, :, p) = \mathbf{R} \odot \mathbf{Q}_p$
end for
$\{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$=PARAFAC ($\mathcal{X}$,$\hat{k}$,non-negativity)
$\hat{\mathbf{R}}$=$\mathbf{AB}^T$
terms=$\emptyset$
for $i = 1 : R$ **do**
  $\hat{\mathbf{r}}_i$ =$\hat{\mathbf{R}}(i, :)$
  Get the corresponding terms of $\hat{\mathbf{r}}_i$
  (e.g all non-zero positions of the vector) and store them in 'w'
  terms=*terms* $\cup$ w
end for
$\underline{\mathbf{T}}$=*terms* $\cap$ *paper_terms*
Construct $\tilde{\mathbf{R}} \in [0, 1]^{(R \times \hat{k})}$ as
$\tilde{\mathbf{R}}$=$\mathbf{R}$(:,indices of '*terms*' in vector '*all_terms*')
if $\tilde{\mathbf{R}}$ contains zero rows **then**
  Construct similarity matrix $\mathbf{S_r}$=$\hat{\mathbf{R}}\hat{\mathbf{R}}^T$
  Set diagonal elements of $\mathbf{S_r}$ equal to 0.
  for each zero row of $\tilde{\mathbf{R}}$ **do**
    Get the reviewer most similar (with non-empty profile) to the one with
    the empty profile and copy the non empty profile
    to the empty one.
  end for
end if
Construct $\tilde{\mathbf{P}} \in [0, 1]^{(P \times \hat{k})}$ as
$\tilde{\mathbf{P}}$=$\mathbf{P}$(:,indices of '*terms*' in vector '*all_terms*')
if $\tilde{\mathbf{P}}$ contains zero rows **then**
  Construct similarity matrix $\mathbf{S_p}$=$\mathbf{PP}^T$
  Set diagonal elements of $\mathbf{S_p}$ equal to 0.
  for each zero row of $\tilde{\mathbf{P}}$ **do**
    Get the paper most similar (with non-empty profile) to the one with
    the empty profile and copy the non zero profile
    to the empty one.
  end for
end if

Chapter 7

Results and Conclusions

7.1 Results

For the evaluation of the methods, data from the ICASSP09 conference was used. The available dataset contains the list of the reviewers and the list of submitted papers, accompanied by the name of the lead author for each paper. The total number of reviewers was $R = 72$ and the number of submitted papers was $P = 132$. A key parameter for the evaluation of the results is the total number of raw terms initially extracted in order to build $\mathbf{M}$ and $\mathcal{X}$. We remind the reader that this parameter is symbolized by T. We observed that the results in terms of precision for all three methods exhibited alterations when changing T. In the following sections, we demonstrate graphically the results of each method by plotting the result matrices of each one and quantitatively, by demonstrating precision graphs for each algorithm.

7.1.1 Illustration of Representative Results

In figure 7.1.1, we can see the original data matrix $\mathbf{M}$. The number of total raw terms for this sample matrix is 2553. The same number of raw terms applies on the following example matrices for each of the three methods.

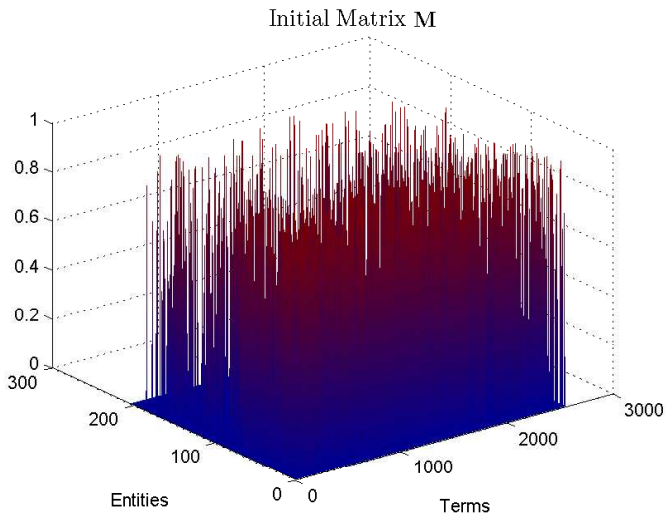
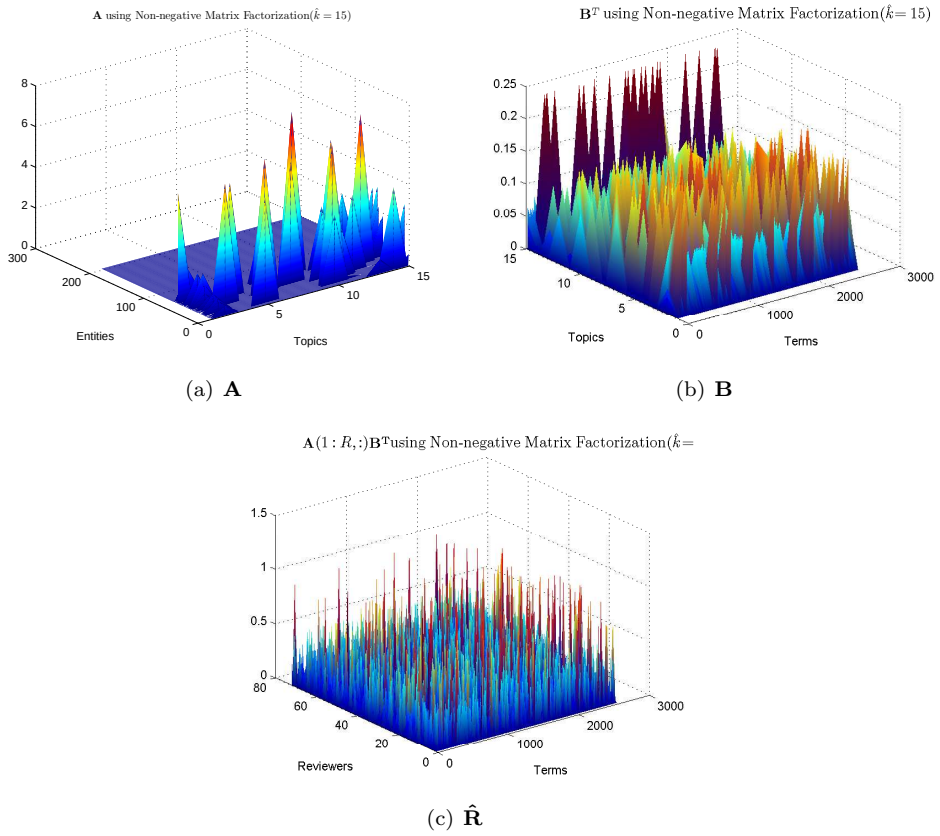


Figure 7.1: Original Data Matrix $\mathbf{M}$

Non-negative Matrix Factorization

The following subfigures in figure 7.1.1 show the factor matrices $\mathbf{A}, \mathbf{B}$ resulting from the Non-negative Matrix Factorization with $\hat{k} = 15$, along with the reconstructed profile matrix $\mathbf{AB}^T$. The size of $\mathbf{AB}^T$ is the same as the size of $\mathbf{M}$. The reviewer profile matrix $\hat{\mathbf{R}}$ can be obtained from the reconstructed matrix by taking the first R rows. The matrix $\hat{\mathbf{R}}$ is shown on the figure below.

In figure 7.1.1, we can see the most significant property introduced by NMF, the non-negativity, which is imposed on both factors. We can also observe that the resulting reconstructed matrix is not at all sparse, at least by inspection.



(a) $\mathbf{A}$ (b) $\mathbf{B}$

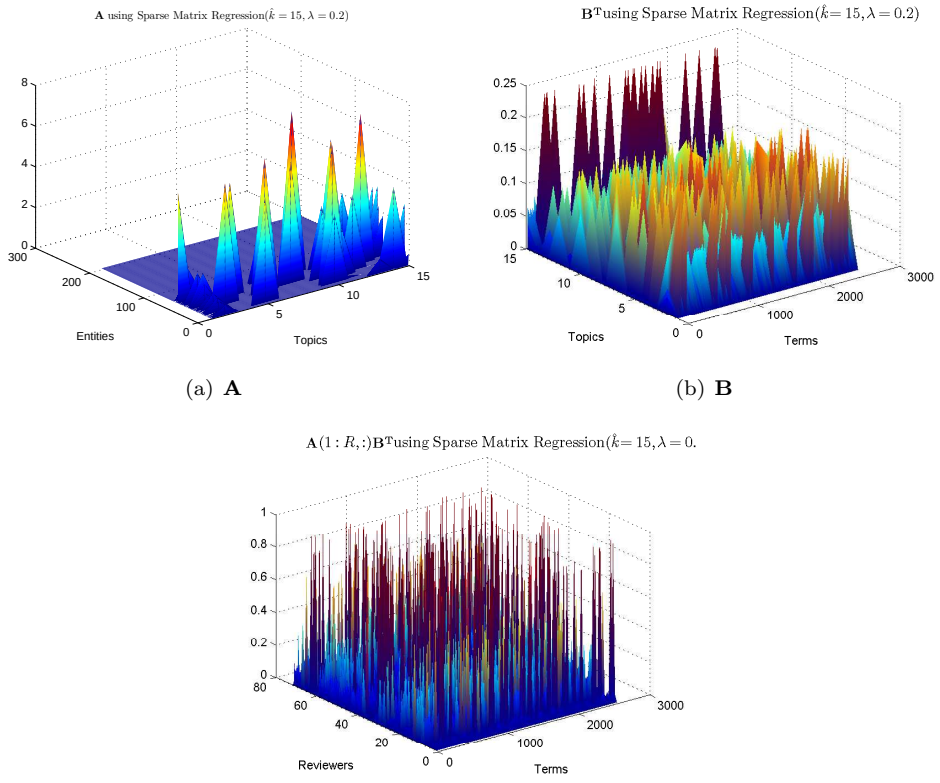
(c) $\hat{\mathbf{R}}$

Figure 7.2: Resulting matrices for Non-negative Matrix Factorization with $\hat{k} = 15$

Sparse Matrix Regression

In figure 7.1.1, we can view the factor matrices $\mathbf{A}, \mathbf{B}$ obtained by Alternating Sparse Matrix Regression with $\hat{k}, \lambda = 0.2$ and the resulting reconstructed matrix $\mathbf{AB}^T$. The same as the NMF case applies as to the computation of $\hat{\mathbf{R}}$. This matrix is shown on the figure below.

The key observation that comes from figure 7.1.1 is the fact that both the factor matrices and the resulting reconstructed matrix are sparse and the resulting matrix is also more sparse than the initial matrix $\mathbf{M}$.



(a) **A** (b) **B**

(c) $\hat{\mathbf{R}}$

Figure 7.3: Resulting matrices for Sparse Matrix Regression with $\hat{k} = 15, \lambda = 0.2$

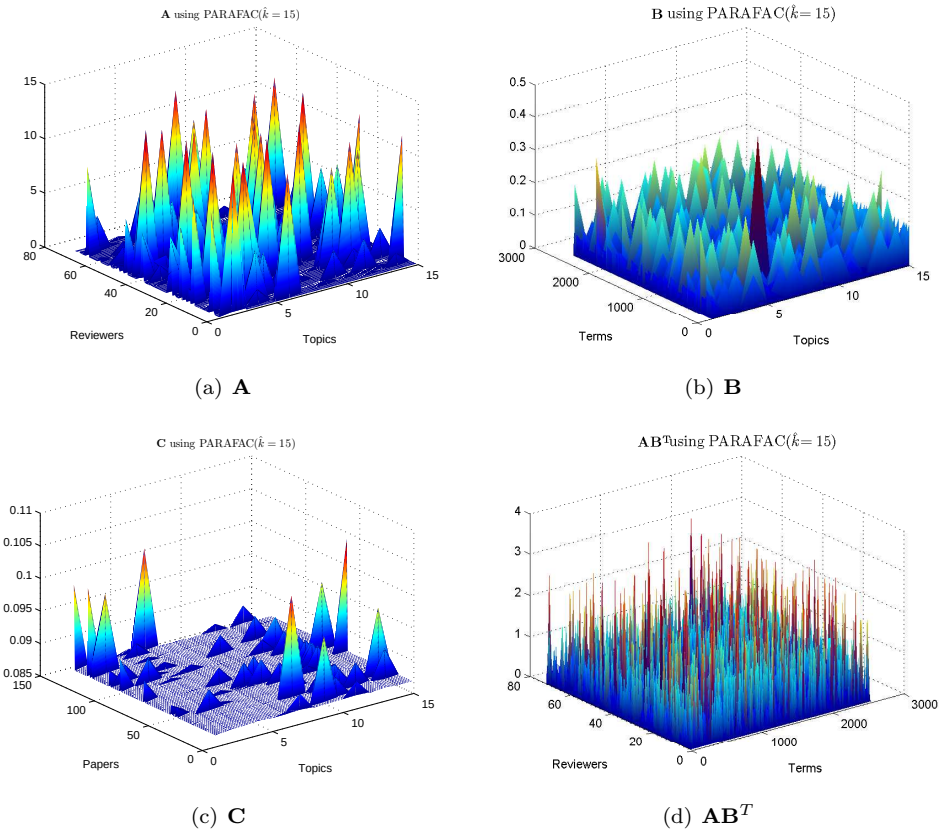
The most severe obstacle that we faced throughout the evaluation of the SMR method was the fact that under certain circumstances, the SMR algorithm produced profiles matrices that contained empty profiles. The main observation upon this fact was that if a profile vector was initially very sparse, after the application of sparse regression, the corresponding reconstructed profile vector would be empty with very high probability. This obstacle was tackled with two methods. The more systematic one was described in the previous chapter, in algorithm 6.5, where we show a Monte Carlo simulation procedure that helps us choose $\{\hat{k}, \lambda\}$ in an appropriate way such that no empty profiles exist. The second, rather empirical, method was the tuning of sparsity in the initial profiles. We observed that when we forced to extract 25-30 raw terms per reviewer, the initial profile vectors were not sufficiently sparse, in order for an empty profile to be produced by SMR.

PARAFAC

In figure 7.1.1, we can view the three modes of the PARAFAC decomposition $[\mathbf{A}, \mathbf{B}, \mathbf{C}]$, as long as the reconstructed matrix $\hat{\mathbf{R}} = \mathbf{A}\mathbf{B}^T$. In the factor matrices, there exist the values of each reviewer, term and paper with respect to the $\hat{k}$ extracted topics. In the following example, $\hat{k}$ is equal to 15.

7.1.2 Quantitative Evaluation

For the needs of evaluating the results of the proposed methods, we employ a familiar and widely used in text mining applications evaluation method, the Precision-Recall metrics. In order to proceed, first we must provide a brief introduction to the basics of this method.



(a) **A** (b) **B**

(c) **C** (d) $\mathbf{AB}^T$

Figure 7.4: Resulting matrices for PARAFAC with $\hat{k} = 15$

Precision is the fraction of the retrieved terms that are also relevant. The precision value for a set of retrieved and relevant terms can be calculated by the following formula:

$$\text{Precision} = \frac{|\text{Relevant} \cap \text{Retrieved}|}{|\text{Retrieved}|} = Pr\{\text{Relevant}|\text{Retrieved}\} \tag{7.1}$$

Recall is the fraction of relevant terms that are retrieved. Recall can be calculated as follows:

$$\text{Recall} = \frac{|\text{Relevant} \cap \text{Retrieved}|}{|\text{Relevant}|} = Pr\{\text{Retrieved}|\text{Relevant}\} \tag{7.2}$$

In order to single out the relevant terms from all the retrieved terms, we employed empirical evaluation. We asked a domain expert to rate each one of the retrieved terms as relevant or not, producing the set of relevant terms that was used for the calculation of precision-recall values and the interpolated curve, as introduced on the previous paragraph.

We must note that the set of the relevant terms for each result set is a subset of the retrieved terms for that set. Therefore, as implied by equation 7.2, the recall value for each of the result sets is equal to 1. For that reason, we only present the precision value, given by equation 7.1. For each method, we provide the precision value for 3 different sets of initial terms, for a range of approximation ranks. These sets are:

- 10 terms per reviewer & 5 terms per paper, resulting to $T = 1251$.

- 20 terms per reviewer & 5 terms per paper, resulting to $T = 1844$.
- 30 terms per reviewer & 5 terms per paper, resulting to $T = 2431$.

For the SMR algorithm specifically, we present the precision in terms of both the approximation rank and the parameter λ that affects sparsity. For the SMR and PARAFAC algorithms, we present an additional metric, the average non-zero element count per row, a number that indicates the average number of terms extracted for each reviewer. For convenience, we symbolize this number as $\overline{\neq 0}$. This number can be seen as a measure of sparsity.

Precision for Non-Negative Matrix Factorization

In the following tables, we present the precision values for 3 different values of $\hat{k}$ (the approximating rank). In the post-processing procedure, we choose $\hat{t} = 15$ highest scoring terms for each reviewer.

$T = 1251$

	$\hat{k} = 20$	$\hat{k} = 30$	$\hat{k} = 40$
precision	0.58065	0.51613	0.47170

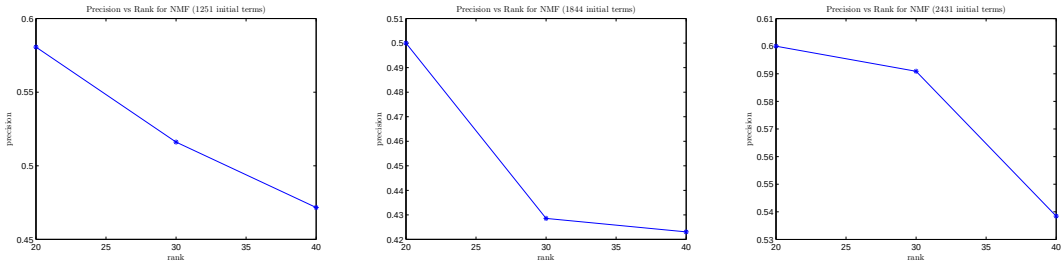
$T = 1844$

	$\hat{k} = 20$	$\hat{k} = 30$	$\hat{k} = 40$
precision	0.5	0.42857	0.42309

$T = 2431$

	$\hat{k} = 20$	$\hat{k} = 30$	$\hat{k} = 40$
precision	0.6	0.59091	0.53846

In the figure 7.1.2, we demonstrate the precision value versus the approximation rank for the NMF algorithm. The rank ranges from 20 to 40. We can observe that the algorithm performs well for low ranks. As the rank grows, the performance of the algorithm drops in terms of precision.



(a) $T = 1251$ (b) $T = 1844$ (c) $T = 2431$

Figure 7.5: Precision vs Rank for Non-Negative Matrix Factorization

Precision for Sparse Matrix Regression

In the following tables, we present the precision values for 3 different values of $\hat{k}$ and λ. For each dataset, we choose a different range of values for λ. This is done because as the initial dimension of the data matrix increases, the density of the profiles grows. So, in order to ensure that the resulting profiles are sufficiently sparse, as the dimension of the data matrix increases, we user higher values for λ.

$T = 1251$

		$\hat{k} = 20$	$\hat{k} = 30$	$\hat{k} = 40$
$\lambda = 0.1$	precision	0.34615	0.30189	0.31373
	$\overline{\neq 0}$	51	44	45
$\lambda = 0.3$	precision	0.44731	0.35714	0.35714
	$\overline{\neq 0}$	32	30	31
$\lambda = 0.6$	precision	0.84615	0.64	0.64
	$\overline{\neq 0}$	19	19	21
$\lambda = 0.9$	precision	0.8	0.8	0.8
	$\overline{\neq 0}$	10	11	12

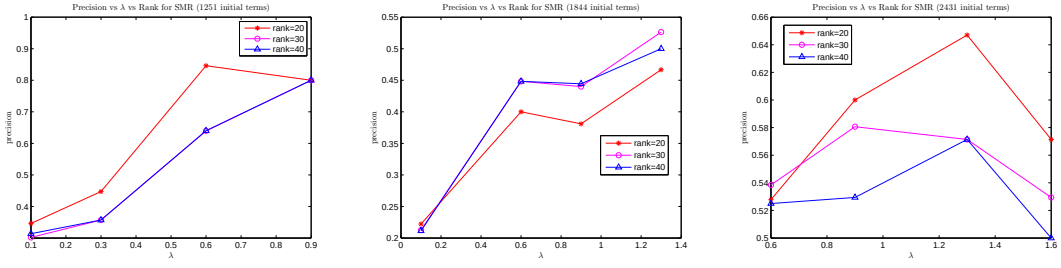
$T = 1844$

		$\hat{k} = 20$	$\hat{k} = 30$	$\hat{k} = 40$
$\lambda = 0.1$	precision	0.22222	0.21260	0.21138
	$\overline{\neq 0}$	225	198	160
$\lambda = 0.6$	precision	0.4	0.44828	0.44828
	$\overline{\neq 0}$	70	66	50
$\lambda = 0.9$	precision	0.38095	0.44	0.44444
	$\overline{\neq 0}$	50	47	49
$\lambda = 1.3$	precision	0.46667	0.52632	0.5
	$\overline{\neq 0}$	18	20	21

$T = 2431$

		$\hat{k} = 20$	$\hat{k} = 30$	$\hat{k} = 40$
$\lambda = 0.6$	precision	0.52778	0.53846	0.525
	$\overline{\neq 0}$	100	93	82
$\lambda = 0.9$	precision	0.6	0.58065	0.52941
	$\overline{\neq 0}$	51	57	47
$\lambda = 1.3$	precision	0.64706	0.57143	0.57143
	$\overline{\neq 0}$	28	30	31
$\lambda = 1.6$	precision	0.57143	0.52941	0.5
	$\overline{\neq 0}$	19	23	26

In the figure 7.1.2, we slightly alter our figure scheme. We demonstrate the evolution of the precision versus the evolution of λ, for a range of rank values.



(a) $T = 1251$ (b) $T = 1844$ (c) $T = 2431$

Figure 7.6: Precision vs λ vs Rank for Sparse Matrix Regression

For all three scenarios, we observe that as λ increases (equivalently, as the resulting profiles become more sparse), the performance, in terms of precision increases too, most of the times, especially for low ranks. This indicates that our insight that sparsity is needed due to the sparse nature of a reviewer's profile, is indeed correct and it's application pays off in terms of performance.

Precision for PARAFAC

In the following tables, we present the precision values for 3 different values of $\hat{k}$ (the number of factors for the PARAFAC model). We must note that for initial dimension 2431 (2431 initial terms), we failed to obtain the PARAFAC decomposition for $\hat{k} = 30$ and $\hat{k} = 40$, due to hardware restrictions. However, the rest of the results suffices for our conclusions.

$T = 1251$

	$\hat{k} = 20$	$\hat{k} = 30$	$\hat{k} = 40$
precision	0.73684	0.68182	0.73913
$\overline{\neq 0}$	90	55	46

$T = 1844$

	$\hat{k} = 20$	$\hat{k} = 30$	$\hat{k} = 40$
precision	0.44828	0.43333	0.43333
$\overline{\neq 0}$	248	270	264

$T = 2431$

	$\hat{k} = 20$	$\hat{k} = 30$	$\hat{k} = 40$
precision	0.51220	-	-
$\overline{\neq 0}$	700	-	-

In the following tables, we include results of the PARAFAC algorithm, using post-processing with $\hat{t} = 15$ highest scoring terms per reviewer. This is is done for the last 2 datasets, where the resulting profiles are not sufficiently sparse.

$T = 1844$

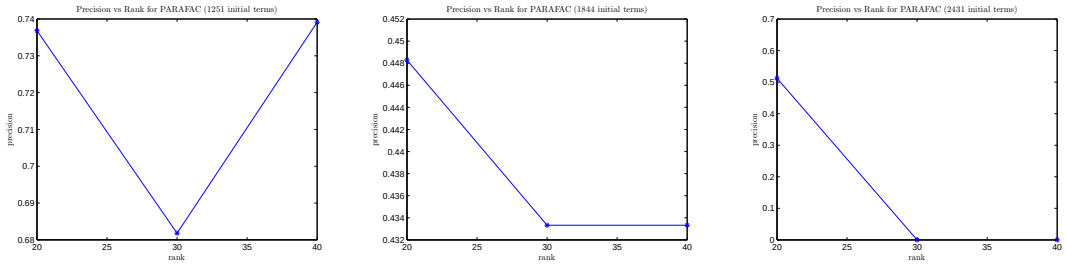
	$\hat{k} = 20$	$\hat{k} = 30$	$\hat{k} = 40$
precision	0.43333	0.43333	0.43333

$T = 2431$

	$\hat{k} = 20$	$\hat{k} = 30$	$\hat{k} = 40$
precision	0.51220	-	-

We can observe that when employing post-processing (in the form of sorting the result vectors), the performance of the PARAFAC based algorithm does not differ significantly from the algorithm without post-processing.

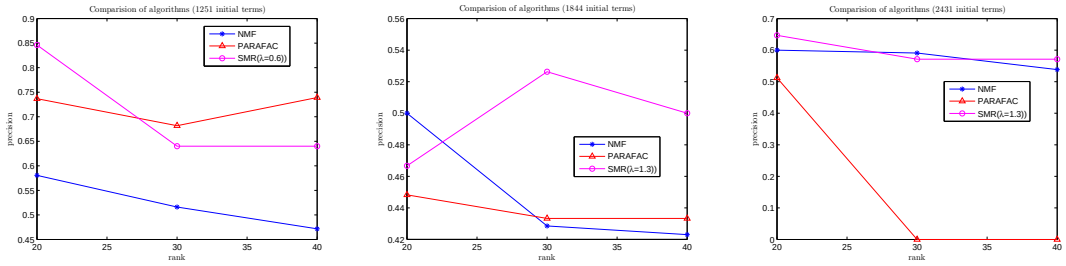
In the figure 7.1.2, we demonstrate the precision value versus the number of trilinear components for the PARAFAC algorithm. The rank ranges from 20 to 40. As observed on the NMF case, the performance of the algorithm drops as the rank increases.



(a) $T = 1251$ (b) $T = 1844$ (c) $T = 2431$

Figure 7.7: Precision vs Rank for PARAFAC

7.2 Discussion and Conclusions



(a) $T = 1251$ (b) $T = 1844$ (c) $T = 2431$

Figure 7.8: Comparison of NMF, PAFAFAC and SMR in terms of precision

In figure 7.2, we can clearly see that in all three figures, there is a choice of parameters that forces Sparse Matrix Regression to outperform both other algorithms. We also observe that for low values of λ, SMR tends to perform similarly to (or even worse than) Non-Negative Matrix Factorization. This result is natural, as low values of λ don't produce sufficiently sparse results.

When it comes to PARAFAC, we can see that for low values of T, the algorithm's performance is better than NMF. When we proceed to higher values of T, due to the data matrix being more dense, the NMF algorithm gains an advantage compared to PARAFAC. When PARAFAC produces sufficiently sparse profile vectors, it's performance is clearly better than NMF (and comparable to the SMR's performance). However, when the resulting profile vectors are dense (like in the cases of 1844 and 2431 initial terms), the need for sparsity rises. This sparsity property can either be achieved artificially by employing post-processing similar to the NMF based algorithm. However, as we shortly describe on the Future Work (7.3) section, there is an alternative by the name of SPARAFAC.

In the following table, we present the computational costs per iteration for each of the three proposed algorithms. Because all three algorithms differ only on the approach of the low dimensional approximation, we focus on the core algorithms, rather than the pre- and post-processing of the data.

NMF	SMR	PARAFAC
$\mathcal{O}(RP\hat{k})$	$\mathcal{O}(RP\hat{k}^2)$	$\mathcal{O}(RPT\hat{k})$

NMF maintains the lowest complexity per iteration. However, because $\hat{k}$ is relatively low in this application, the difference between NMF and SMR in practice, is slightly noticeable and increases as $\hat{k}$ increases. The PARAFAC decomposition is by far the most costly; PARAFAC is the only algorithm among the three with a complexity depending on T. As we presented earlier, T ranges from ≈ 1000 to ≈ 2000 and perhaps more, resulting to very high computational cost for our PARAFAC based approach.

In conclusion, and computational costs aside, we can safely infer that Sparse Matrix Regression produces the highest scoring results. In the case of $T = 1844$, where the resulting profiles of PARAFAC are sufficiently sparse, the algorithm's performance is still clearly higher than NMF's. This means that the two newly proposed methods in this thesis, generally offer better results than the NMF algorithm, a widely used and popular principle in various Data Mining applications. In fact, the algorithm proposed in Algorithm 6.2, offers the Non-negative Matrix Factorization paradigm an advantage, as it substitutes it's fundamental lack of sparsity with the post-processing of the results.

7.3 Future Work

The problem this thesis addresses is widely exploratory and interdisciplinary. For that reason, there is a number of alternatives or expansions that could be made upon this thesis. Some proposals are:

- Employ a newly introduced Three way method called SPARAFAC. SPARAFAC uses the PARAFAC decomposition in order to compute the factor matrices $[\mathbf{A}, \mathbf{B}, \mathbf{C}]$ of the trilinear decomposition and then, uses Sparse Matrix Regression in an alternating fashion, to refine the factor matrices, in terms of sparsity.
- Use a set of terms, predefined by an authority (e.g. the TPC chair), as a filter, in order to refine the initial extraction of the T index terms.
- Extend GoogleScholarMiner, in order to perform citation analysis and eliminate self-citations from a citation count.

Appendix A

Groups of terms for ICASSP 2009 using SMR profiling

In this appendix we present an example of extracted topics using SMR. As we reviewed earlier, the columns of the left factor matrix $\mathbf{B}$ define the basis vectors for the $\hat{k}$-dimensional latent term space. Each one of these compound vectors represents an entire category of terms that constitute a topic

The following list consists of the most notable term groups, as extracted using SMR. The selection of the groups was again made by a domain expert.

Sparse Matrix Regression was chosen for this demonstration due to the sparsity imposed on $\mathbf{B}$. Sparsity on $\mathbf{B}$ implies that each column of $\mathbf{B}$ has a limited number of non-zero values, thus making it easier to visualize it as a group of affine terms.

The choice of parameters for the following example is $\hat{k} = 20$, $\lambda = 0.6$ and $T = 1251$ (one might notice that this choice of parameters is the one that produced the highest precision value among all data sets). The dataset used comes from ICASSP 2009 and is the same dataset used on chapter 7.

Group 1
- blind
- filter banks
- impulse noise
- ofdm
- oversampled filter
- postfix
- postfix ofdm
- real bch
- source channel
- watermarking

Group 2
- opportunistic spectrum
- sensor
- sensor networks
- spectrum access

Group 3
- 2k
- asynchronous cooperative
- block codes
- codes
- complex orthogonal
- orthogonal
- orthogonal space-time
- space-time
- space-time block
- space-time trellis

Group 4
- blind

Group 5
- access control-physical
- beamforming
- downlink beamforming
- greedy user
- simple new

Group 6
- filter banks
- hyperbolic
- hyperbolic class
- nonstationary
- oversampled
- oversampled filter
- quadratic time-frequency
- time-frequency
- time-frequency representations
- underspread

Group 7
- block codes
- mimo
- space-time block

Group 8
- turbo

Group 9
- tone
- tone equalization

Group 10
- multi-hop

Group 11
- coherence
- coherence source
- cramer-rao
- cramer-rao bound
- distributed sensor
- non-gaussian
- partial spatial
- sensor arrays
- source localization
- spatial coherence

Group 12
- cooperative
- cooperative broadcast
- cooperative communications
- cooperative transmission
- dense wireless
- filterbank
- multipath
- precoders
- unknown multipath
- wireless networks

Appendix B

Reviewer Assignment for SPAWC 2010

For the purpose of evaluating qualitatively our SMR & PARAFAC algorithms, we used the profiles produced as input to the algorithm developed in [8], an application that deals with the Reviewer-Assignment Problem using optimization methods.

The sets of reviewers and submitted papers come for the IEEE SPAWC 2010 conference. We conducted an evaluation prior to the submission deadline and one after.

Let us now consider the case of a random reviewer assignment. Without loss of generality, let us assume that each assignment consists of 4 papers per reviewer. Now, let us assume that each reviewer's expertise covers $\frac{1}{7}$th of the broad scientific field of the conference. For this random assignments, the probability of a *bad* paper assignment to a reviewer is equal to the probability that 3 out of the 4 assigned papers are not suitable plus the probability that all 4 assigned papers are bad choices:

$$Pr\{\text{bad}\} = \binom{4}{3}\frac{1}{7}\left(\frac{6}{7}\right)^3 + \left(\frac{6}{7}\right)^4 \approx 0.9$$

A *good* assignment is an assignment the TPC chair would normally do manually, where a *very good* assignment is above the average level a reviewer would expect from the TPC chair. For the subject datasets we present the number of very good, good and bad assignments, along with the probability of a bad assignment. This evaluation was made by a domain expert who has also served as a TPC chair in the past.

Dataset obtained after the submission deadline

The total number of submitted paper was 203 and the number of registered reviewers was 64. We conducted an experiment for both SMR & PARAFAC algorithms, using different numbers of initially extracted terms (due to memory restrictions of the PARAFAC decomposition).

- **SMR**: The dimension of the term space was $T = 3113$, $\lambda = 0.9$ and $\hat{k} = 20$. The grading of the resulting assignments is:
  Very Good=20, Good=32, Bad=12, $Pr\{bad\} = 0.1875$.
- **PARAFAC**: The dimension of the term space was $T = 1676$ and $\hat{k} = 20$. The dimension T was chosen significantly lower than above, due to complexity restrictions. The grading of the resulting assignments is:
  Very Good=17, Good=32, Bad=15, $Pr\{bad\} = 0.2344$.

We observe that SMR performs slightly better than PARAFAC. The degradation of PARAFAC's performance could be partially due to the limited number of initially extracted terms.

Dataset obtained prior to the submission deadline

The number of submitted papers was 78 and the number of registered reviewers was 64. The dimension of terms was $T = 2288$. We used SMR with $\lambda = 0.1$ and $\hat{k} = 20$. The number of very good assignments was 24, the number of good assignments was 22 and the number of bad assignments was 18. This resulted to $Pr\{bad\} = 0.2812$.

Conclusions

At the same time we conducted the above experiments, we also produced an assignment using custom profiles manually selected by authors and reviewers. The probability of a bad assignment in this manual reviewing assignment was 0.047. This assignment requires participation not only from the TPC chair but from reviewers and authors too. Reviewers and authors were requested to provide manually profiles for themselves or their paper. In a fully manual assignment by the TPC chair, the number of bad assignments would be around 7 out of 64; this however, would require possibly a week of hard work from the chair. On the other hand, our automated approach, involving zero human work, offers considerably good assignments. The fully automatic results may be worse than both manual assignments but considerably better than a random assignment (where 50 out of 64 assignments are bad); the automatic assignments can be further improved by the TPC chair by a less than a day's work of swapping and refining assignments.

Manual	**Custom profiles**	**Automatic**	**Random**
TPC chair: 7 days	TPC chair: 2-4 hrs	TPC chair: 0 hrs	TPC chair 10 min
Reviewer: 0 hrs	Reviewer: 2 min	Reviewer: 0 hrs	Reviewer: 0 hrs
Author: 0 hrs	Author: 2 min	Author: 0 hrs	Author: 0 hrs
$Pr\{bad\} = 0.109$	$Pr\{bad\} = 0.047$	$Pr\{bad\} = 0.1875$	$Pr\{bad\} = 0.9$

Furthermore, the final reviewer profiles depend highly on the papers submitted to the conference. This should be fairly obvious to the reader, because the proposed algorithms treat reviewer and paper profiles jointly. For example, if a reviewer is an expert at OFDM but none of the submitted papers is regarding this subject, that reviewer will not be matched by the term 'OFDM' in the final profile.

In addition to the latter remark, we observed that reviewers with very common last names generally suffer from non-representative profiles. This disambiguation issue can be fixed by human intervention to the profiling process.

Finally, an important observation made when evaluating the resulting assignments is that each one of the final terms can be seen as the most representative term of it's corresponding topic group (see also Appendix A); relationships between term groups are preserved in the low dimensional space, therefore, even though some profile terms may seem quite limited, there is a good chance they represent a larger group of similar/relevant terms.

Bibliography

[1] G. Giannakis, P. Stoica, and Y. Hua, *Signal Processing Advances in Wireless and Mobile Communications, Volume 2: Trends in Single-and Multi-User Systems*. Prentice Hall PTR Upper Saddle River, NJ, USA, 2000. v, 12, 27

[2] C. Faloutsos, T. Kolda, and J. Sun, "Mining large time-evolving data using matrix and tensor tools," in *Int. Conf. on Data Mining*, Citeseer, 2007. v, 15

[3] D. Lee and H. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, 1999. v, 16, 17, 18

[4] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996. v, 22, 24

[5] T. Kolda and B. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009. v, 27, 29

[6] T. Kolda and B. Bader, "The TOPHITS model for higher-order web link analysis," in *Workshop on Link Analysis, Counterterrorism and Security*, vol. 7, pp. 26–29, Citeseer, 2006. v, 32, 33, 46

[7] B. Bader, M. Berry, and M. Browne, "Discussion tracking in Enron email using PARAFAC," *Survey of Text Mining II: Clustering, Classification, and Retrieval*, pp. 147–163. v, 32, 33, 34

[8] N. Sidiropoulos, "Conference program optimization as vector quantization under session capacity constraints, (unpublished working paper)," 2009. 2, 63

[9] F. Wang, B. Chen, and Z. Miao, "A survey on reviewer assignment problem," *Lecture Notes in Computer Science*, vol. 5027, pp. 718–727, 2008. 2

[10] K. Frantzi, S. Ananiadou, and H. Mima, "Automatic recognition of multi-word terms:. the C-value/NC-value method," *International Journal on Digital Libraries*, vol. 3, no. 2, pp. 115–130, 2000. 7

[11] I. Witten, G. Paynter, E. Frank, C. Gutwin, and C. Nevill-Manning, "KEA: Practical automatic keyphrase extraction," in *Proceedings of the fourth ACM conference on Digital libraries*, pp. 254–255, ACM New York, NY, USA, 1999. 8

[12] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American society for information science*, vol. 41, no. 6, pp. 391–407, 1990. 13

[13] J. Kleinberg, "Authoritative sources in a hyperlinked environment," *Journal of the ACM (JACM)*, vol. 46, no. 5, pp. 604–632, 1999. 14

[14] T. Kolda, "Multilinear algebra for analyzing data with multiple linkages." `http://www.telecom.tuc.gr/~nikos/TRICAP2006main/Kolda-TRICAP-June2006.ppt`, 2006. 14

[15] N. Origins, "Nonnegative Matrix and Tensor Factorizations for Text Mining Applications," *children*, vol. 5, no. 4, p. 3. 16, 17

[16] C. Boutsidis and E. Gallopoulos, "SVD based initialization: A head start for nonnegative matrix factorization," *Pattern Recognition*, 2007. 17

[17] F. Shahnaz, M. Berry, V. Pauca, and R. Plemmons, "Document clustering using nonnegative matrix factorization," *Information Processing and Management*, vol. 42, no. 2, pp. 373–386, 2006. 17

[18] E. Allan, M. Horvath, C. Kopek, B. Lamb, T. Whaples, and M. Berry, "Anomaly Detection Using Nonnegative Matrix Factorization," *Survey of text mining II: clustering, classification, and retrieval*, p. 203, 2007. 17

[19] A. Cichocki, R. Zdunek, A. Phan, and S. Amari, *Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-way Data Analysis and Blind Source Separation*. Wiley, 2009. 17

[20] D. Donoho and V. Stodden, "When does non-negative matrix factorization give a correct decomposition into parts," *Advances in neural information processing systems*, vol. 16, pp. 1141–1148, 2004. 19

[21] L. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, 1966. 29

[22] J. Kruskal, "Three-way arrays: Rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics," *Linear Algebra Appl*, vol. 18, no. 2, pp. 95–138, 1977. 31

[23] N. Sidiropoulos and R. Bro, "On the uniqueness of multilinear decomposition of N-way arrays," *Journal of chemometrics*, vol. 14, no. 3, pp. 229–239, 2000. 31

[24] R. Harshman, "PARAFAC2: Mathematical and technical notes," *UCLA working papers in phonetics*, vol. 22, pp. 30–47, 1972. 32

[25] J. Douglas Carroll, S. Pruzansky, and J. Kruskal, "CANDELINC: A general approach to multidimensional analysis of many-way arrays with linear constraints on parameters," *Psychometrika*, vol. 45, no. 1, pp. 3–24, 1980. 32

[26] R. Harshman and M. Lundy, "Uniqueness proof for a family of models sharing features of Tucker's three-mode factor analysis and PARAFAC/CANDECOMP," *Psychometrika*, vol. 61, no. 1, pp. 133–154, 1996. 32

[27] J. Carroll and J. Chang, "Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decomposition," *Psychometrika*, vol. 35, no. 3, pp. 283–319, 1970. 32

[28] B. Bader and T. Kolda, "Matlab tensor toolbox version 2.3." `http://csmr.ca.sandia.gov/~tgkolda/TensorToolbox/`, 2009. 33

[29] B. Bader and T. Kolda, "Efficient MATLAB computations with sparse and factored tensors," *SIAM Journal on Scientific Computing*, vol. 30, no. 1, pp. 205–231, 2007. 33

[30] B. Bader and T. Kolda, "Matlab tensor classes for fast algorithm prototyping," *ACM Trans. Math. Software*, vol. 32, pp. 635–653, 2006. 33

[31] C. Andersson and R. Bro, "The N-way Toolbox for MATLAB." `http://www.models.life.ku.dk/source/nwaytoolbox/`, 2000. 33, 48

[32] "Google scholar." `http://scholar.google.com/`. 35

[33] "Publish or perish." `http://www.harzing.com/pop.htm`. 35

[34] Stanford, "Part of speech tagger." `http://nlp.stanford.edu/software/tagger.shtml`. 37

[35] J. Sun, H. Zeng, H. Liu, Y. Lu, and Z. Chen, "CubeSVD: a novel approach to personalized Web search," in *Proceedings of the 14th international conference on World Wide Web*, p. 390, ACM, 2005. 46

[36] N. Liu, B. Zhang, J. Yan, Z. Chen, W. Liu, F. Bai, and L. Chien, "Text representation: From vector to tensor," in *Proceedings of the Fifth IEEE International Conference on Data Mining*, pp. 725–728, IEEE Computer Society, 2005. 46

[37] A. Mirzal, "Weblog Clustering in Multilinear Algebra Perspective," *International Journal of Information Technology*, vol. 15, no. 1, 2009. 46