Textual Influence Modeling Through Non-Negative Tensor Decomposition

A Dissertation Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Robert Earl Lowe

August 2018

For my wife, Erin

Acknowledgments

I would like to thank my advisor, Dr. Michael Berry, for his guidance and encouragement. I would also like to thank my committee members Dr. Judy Day, Dr. Bradley Vander Zanden, and Dr. Audris Mockus for all of their helpful input. Thank you to Ms. Dana Bryson, without whom the paperwork surrounding the completion of this dissertation would have simply not been possible. Last, but certainly not least, I would like to thank my wife, Erin Lowe. Without Erin's encouragement, I would have never gone to graduate school and without her love and support I certainly would have never succeeded.

Abstract

No document is created in a vacuum. In all literature, there exists some influencing factor either in the form of cited documents, collaboration, or documents which authors have read. This influence can be seen within their works, and is present as a latent variable. This dissertation introduces a novel method for quantifying these influences and representing them in a semantically understandable fashion. The model is constructed by representing documents as tensors, decomposing them into a set of factors, and then searching the corpus factors for similarity.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

> *Nam cum pictor praecogitat quae facturus est, habet quidem in intellectu sed nondum intelligit esse quod nondum fecit.*
>
> – Anselm of Canterbury [7]

In the eleventh century, Anselm of Canterbury wrote what has since come to be known as the ontological argument for the existence of God [7]. Anselm's argument was based on the assumption that all ideas, or more specifically, all thoughts originate either from perceptions of the outside world or from images formed within the imagination. From this he provides an argument for the existence of a divine being. The research presented here follows this same epistemological assumption to a much less trivial end. Instead of proving divine influence, the present work shall attempt to measure the influence present in the written works of less divine beings.

The basic assumption made about text documents is the same assumption that Anselm made about the origin of thoughts. Every word, phrase, sentence, paragraph, and theme in a document must come from one of two sources. Either the author created the thought from within their own mind, and as such this counts as a literary contribution, or the author could have transferred ideas from some outside source. These sources can take on many forms. In the case of academic writing, the author is likely to have been influenced primarily by the various books and papers that they have read over the course of their research. Another form of influence is a coauthor (though in the case of academic literature, coauthors are almost always explicitly stated.) Of course, the influence over the text in a paper is not constrained merely to the literature that the author has cited, but is ultimately a reflection of an author's entire life experience and background. In the case of literary writing, such as a novel or play, a reasonable assumption is that an author is influenced by other works within their genre as well as by the society in which they live.

Given that every written document is influenced by at least a small set of outside documents, the present work attempts to model and quantify this influence by separating documents into a set of factors and then searching for common factors among the documents. The desired result has two parts. First, a weight is assigned to each factor indicating its importance in the target work. Second, the factors themselves should carry enough semantic meaning to identify the ideas and elements of style which have been transferred from a source document to a target document. Thus, the goal of the present work is to identify influencing factors and to quantify the influence they exert on a target document.

The usefulness of such a measurement should be readily apparent to anyone working in any academic field. In modern research, the performance of participants is rooted in an attempt to measure that person's influence over their chosen field. Traditional approaches to this problem involve counting citations over a specific window of time [1] while more modern approaches tend to involve some document semantics [11, 16]. Measuring influence in a written document can also be applied in situations where authorship is in question. Given a corpus of works of confirmed provenance, and a disputed document, influence modeling can identify the possible influence of each author. Thus textual influence modeling can be used to answer the question of authorship where it is disputed, or could potentially be used to identify plagiarized passages.

1.1 Modeling Influence

At a high level, an influence model identifies elements that appear to have been incorporated into a target document from a source document. These elements are numerous, and are generally perceived on an intuitive level. For example, they could include elements of style, topics, phrases, or ideas. A human reader seems to be able to identify these elements on an intuitive level, as can be seen readily whenever a reader says one author “sounds like” another. This operation is also in effect when tracing ideas through written academic literature. In either case, the text of a target document along with its corpus of cited documents seems to provide sufficient evidence to identify potential sources of influence in the target document.

The chief problem with an intuitive model such as the one outlined in the previous paragraph is that it is highly subjective. Every conclusion reached by human scholars in such a system must appeal to intuition and logic, and so determining the strength of any perceived relationships present in the corpus presents a difficult challenge. In recent years, the emerging field of computational stylistics has offered several techniques for quantifying these elements of style which can serve as markers of influence [3, 10, 6]. In so much as it can, computational stylistics has the principal goal of using textual evidence to answer the question of authorship. The current state of the art techniques for addressing these questions rely upon statistical analysis of word frequencies within documents [10]. The typical approach is to use a set of “marker words” to determine the likelihood of an author’s contribution to a target document. Those words that are more likely to occur in the works of one author are ascribed to them if there is sufficient statistical significance of the word’s classifying power. This current approach offers only a coarse level of determination. Computational stylistic analysts can identify words that are more likely to come from one author’s work, and they in turn identify whether that author appears to have contributed to a target document. Thus the current techniques only inform the probability of an author’s contribution as a dichotomy. Each potential author was either a contributor, or they were not. The objective of the model outlined in this dissertation is to extend this model to include more detail. As opposed to determining whether an author has contributed to a target work directly, this model assumes that influence is present in multiple forms; the present model seeks to identify the strength of influence, as well as to identify what those specific influences were.

1.1.1 Tensors and Decompositions

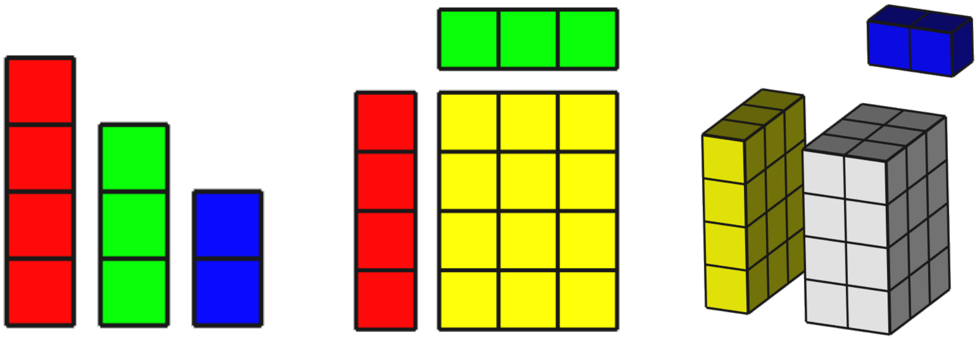
In order to analyze a document, it must first be quantified in some way that allows for analysis. The model discussed in this dissertation represents documents using tensors. The term tensor has been broadly applied across multiple fields to describe several different types of related objects. For the purposes of factor analysis, a tensor is simply an extension of matrices into a higher number of modes. In tensor terminology a "mode" is a dimension along which the tensor can be indexed. A scalar is a mode zero tensor, a vector is a mode one tensor, and a matrix is a mode two tensor. When the number of modes exceeds two, it is customary to refer to the array simply as a tensor. A detailed account of the tensor operations performed by this model is given in the next chapter. For a complete treatment of tensors as they pertain to factor analysis, see Tamara Kolda's tutorial [19].

The underlying principle of tensor analysis is polyadic decomposition, which was first described by Frank Hitchcock in 1927 [15]. When a tensor is expressed in polyadic form, it is expressed as the sum of rank 1 tensors, which is usually written as the outer product of vectors. (This is also referred to as the tensor product of vectors, which is in line with the geometric interpretation of tensors as the outer product of vector spaces.) Each polyadic factor in a tensor of m modes is the tensor product of m vectors. For example, given a 3-mode tensor $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$, its polyadic decomposition into r factors is a set of factors which satisfies Equation 1.1. For the sake of convenience, the remainder of this discussion will assume a three mode tensor, however everything discussed here can readily be extended to any number of modes.

$$\mathcal{T} \approx \sum_{i=1}^{r} a_i \otimes b_i \otimes c_i \tag{1.1}$$

The tensor, or outer, product $a \otimes b$ used in Equation 1.1 results in a tensor where the modes are the concatenation of the modes of a and b. For instance, if a and b are vectors of size i and j respectively, $a \otimes b$ results in a 2-mode tensor with dimensions $i \times j$. In the case of building a 3-mode tensor, three vectors are needed, and the elements of the product result are computed as in in Equation 1.2. A graphical representation of this product is shown in Figure 1.1. Of note is how each vector serves as scaling values for a mode.

$$\mathcal{T}_{ijk} = a_i b_j c_k \tag{1.2}$$



(a) Mode Vectors   **(b)** $a \otimes b$ **(c)** $a \otimes b \otimes c$

Figure 1.1: Tensor Product

The notion of the tensor product also gives rise to the notion of tensor rank. Tensor rank is not the same as matrix rank, and is in fact much more difficult to compute. Perhaps the easiest way to understand the notion of tensor rank is recursive. A rank 1 tensor is a tensor which can be constructed completely from the tensor product of 1-mode tensors (vectors). The tensor described in Equation 1.2 and Figure 1.1 is therefore a rank-1 tensor, as are all the factors in the polyadic decomposition. A tensor of general rank r is the result of the summation of r rank-1 tensors. The rank of a tensor $\mathcal{T}$ is therefore defined as the minimum number of rank-1 tensors needed to sum to $\mathcal{T}$.

Hitchcock's paper mainly presents the polyadic decomposition from a purely mathematical perspective, with applications to studying tensor invariants and tensor rank. In fact, as later papers show, the problem of determining the rank of a tensor is NP-Complete [14]. Polyadic decomposition began to see other uses when it was rediscovered in 1970 by Richard Harshman [12], Douglas Carroll, and Jih-Jie Chang [8]. Harshman coins the term "PARAFAC", a portmanteau of "Parallel Factors" while Carroll and Chang refer to the model as "CANDECOMP" in placed of "Canonical Decomposition". Both papers present the model as a means of studying psychological data by treating the tensor factors as explanatory variables for the variance in the tensor data. In recent years, tensor analysis has begun to take root in other fields such as chemometrics [4] and text mining [3]. In several modern treatments, the polyadic decomposition is referred to as "CPD" or "Canonical Polyadic Decomposition". For the purposes of factor analysis, factors are often normalized, without loss of generality [4, 3], yielding the decomposition shown in Equation 1.3.

$$\mathcal{T} \approx \sum_{i=1}^{r} \lambda_i a'_i \otimes b'_i \otimes c'_i \tag{1.3}$$

Here λ_i is a scalar where $\lambda_i = \text{norm}(a_i \otimes b_i \otimes c_i)$. This is desirable because the factors in this form are proportional profiles [12] with all of the magnitude of the factor contained in λ_i. As can be clearly seen from Equation 1.3, λ_i is also an expression of the influence that factor i exerts over the tensor $\mathcal{T}$. For this reason, factors are typically expressed in order from largest λ_i to the smallest λ_i. Thus these factor norms serve a similar purpose as eigenvalues in principal component analysis, or as singular values in singular value decomposition.

In fact, the similarities between PCA, SVD, and CPD do not end with the inclusion of weights! Nor is it true that CPD is the only tensor decomposition. The closest competing decomposition is the Tucker decomposition, first proposed in 1963 and fully formed in 1966 [19]. Given tensor $\mathcal{T}$, the Tucker decomposition yields the factor matrices $\mathbf{A} \in \mathbb{R}^{I\times P}$, $\mathbf{B} \in \mathbb{R}^{J\times Q}$, and $\mathbf{C} \in \mathbb{R}^{K\times R}$. The model also contains a so-called core tensor $\mathcal{G} \in \mathbb{R}^{P\times Q\times R}$. These factors are fit to satisfy Equation 1.4, where $\mathcal{G} \times_n \mathbf{M}$ is the n-mode product of tensor $\mathcal{G}$ and matrix $\mathbf{M}$.

$$\mathcal{T} \approx \mathcal{G} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C} \tag{1.4}$$

An element-wise version of the tucker decomposition is shown in Equation 1.5. For a complete treatment of the n-mode tensor matrix product, see the Kolda tutorial [19].

$$t_{ijk} \approx \sum_{p=1}^{P}\sum_{q=1}^{Q}\sum_{r=1}^{R} g_{pqr}a_{ip}b_{jq}c_{kr} \tag{1.5}$$

As was shown by Henk Kiers, the relationship between PCA, Tucker decomposition, and CPD is hierarchical [18]. While Kiers's paper focuses on 3-way analysis, his results extend to any number of modes. Because a tensor can be unfolded along any dimension to form a two dimensional matrix, it is always possible to use PCA to find explanatory factors for tensor data. In fact, Tucker-3 is a constrained version of PCA. The exact nature of these constraints is beyond the scope of the present discussion, however they are a direct result of the presence of the core tensor. A simplified summary is that the core tensor's dimensions predetermines the number of factors to be discovered. CPD, in turn, is a constrained variant of the Tucker decomposition. While the CPD is usually written without a core tensor, it can be thought of as having an identity tensor as its core. The identity tensor is simply a tensor with ones along its super-diagonal and zeros everywhere else. (Or stated more formally, an identity tensor is a tensor containing ones where $i_1 = i_2 = \ldots = i_n$ for all n modes and zeros in all other positions.) Also, in CPD, $P = Q = R$ (and so on if there are more than three modes). Tucker decomposition allows for each mode to have a different number of factors, while CPD does not. As can be expected, the more constraints placed upon the explanatory model of the tensor comes at the expense of quality of fit. Hence, PCA will always provide the best fit, Tucker will either be as good or worse than PCA, and CPD will always be as good or worse than a Tucker model [18, 4].

So then if the tensor decomposition models provide a worse fit, the question becomes why are they important? The answer lies in several properties of the tensor decompositions. First, tensor decompositions retain the structure of the original data [12, 19]. Unfolding a tensor into a matrix loses semantic information about the variables being analyzed, and extracting intuitive semantics from the resultant PCA model is difficult and usually impossible [4]. Also, in the case of CPD, the factors are unique under rotation so long as the number of factors extracted is greater than or equal to the rank of the tensor [13]. Another desirable property of the CPD is that it does not partition space by hyper-surfaces. Instead, it creates a sort of implicit set of axes for factor separation by providing a proportional profile along the tensor's basis [13]. Thus if several tensors of like dimensions are decomposed, their factors can be logically thought of as existing within the same space. This allows for comparison among the factors to be carried out, unlike under PCA where the factor space of each matrix is a projection into a new space, making comparison of factors from disparate matrices difficult to perform in a meaningful way.

In some instances of tensor analysis, it can be convenient to apply additional constraints to the model. The most common constraint applied to CPD is a non-negativity constraint [21, 4, 19]. This is done for a variety of reasons, most notably as a form of dimension reduction and when analyzing data which are naturally predisposed to be non-negative. In the model presented in this dissertation, both outcomes are necessary. First, the tensors used in this model are extremely sparse, and so introducing negative factors makes the search space for factors so large that fitting the model becomes intractable. Second, the tensors used in this model represent frequency data, which means that negative values in factors would have no valid semantic meaning.

1.1.2 Representing Documents as Tensors

The text documents to be analyzed are represented as a tensor by dividing them into phrases of length n. These phrases, commonly referred to as n-grams, are counted and their frequencies are entered into a tensor. Each word in the corpus vocabulary is assigned an index, and the tensors n modes refer to these indexes. For example, suppose $n = 3$. The document tensor $\mathcal{D}$ would have 3 modes. The entry d_{ijk} refers to the frequency of the n-gram consisting of words i, j, and k from the corpus vocabulary.

The tensors produced by this encoding will be cubic. Given a vocabulary consisting of v words, the resultant tensor will have v indexes in each mode. The tensor can represent the frequency of all possible n-grams, and as such will be extremely sparse as very few of these n-grams are likely to appear in a document. When decomposed into polyadic form, these tensors will yield sets of related words as well as related n-grams that appear in the same factors.

1.1.3 Modeling Influence

The basic model applied to the document begins with the decomposition of a document into its individual factor tensors.

$$\mathcal{D} = \sum \mathcal{F}_i \tag{1.6}$$

where $\mathcal{F}_i \in \mathrm{F}$ is a factor of the tensor $\mathcal{D}$, ($\mathcal{F}_i = a_i \otimes b_i \otimes c_i$). As has been previously noted, the model becomes more expressive by separating out a normalizing value λ_i from each f_i. Thus the decomposed document becomes:

$$\mathcal{D} = \sum \lambda_i \mathcal{F}'_i \tag{1.7}$$

where $\mathcal{F}'_i = \frac{1}{|\mathcal{F}_i|}\mathcal{F}_i$ and $\lambda_i = |\mathcal{F}_i|$. This is desirable for two reasons. Given that all tensors within the corpus have the same dimensions, and that all are decomposed using Non-Negative CPD, the factors occupy the same type of space as the factors of other documents. By normalizing them into a proportional model of the document, the factors become directly comparable irrespective of the magnitude of influence they exert in their source document.

Let C be a corpus of documents, encoded as tensors $\mathcal{D}_j \in \mathrm{C}$. Let $\mathcal{D}_t$ be the target document to be studied, and all other documents in $\mathrm{S} = \mathrm{C} - \mathcal{D}_t$ are treated as source documents for $\mathcal{D}_t$. The goal of the influence model is to ascribe the factors of $\mathcal{D}_t$ to a factor from each source document $\mathcal{D}_s \in$ S and assign weights to each of the source document influences. Each document in C is decomposed as per Equation 1.7. $\mathcal{D}_t$ is also decomposed into its components. This produces sets of factors $\mathrm{F'}_s$ and Λ_s for each source document as well as $\mathrm{F'}_t$ and Λ_t for the target document. By measuring the similarity of each factor $f'_t \in \mathrm{F'}_t$ and source factors $f'_s \in \mathrm{F'}_s$ in every $\mathrm{F'}_s$, each factor can be ascribed to a source document $\mathcal{D}_s$ or as being original to $\mathcal{D}_t$. Using the similarity measurements between the f'_t and f'_s factors, each corresponding f_t factor of $\mathcal{D}_t$ is categorized as either belonging to one of several sets: F_t^s for all factors of $\mathcal{D}_t$ ascribed to some factor of $\mathcal{D}_s$ and F_t^n for all factors with no matching source.

For each of these factor sets, tensors can be formed by summing over the set. For every F_t^s, the tensor $\mathcal{F}_t^s$ is the sum of all components of document t ascribed to document s.

Hence, the model of the target document can be rewritten:

$$\mathcal{D}_t \approx \sum_{s=1}^{|\mathrm{S}|} \mathcal{F}_t^s + \mathcal{F}_t^n \tag{1.8}$$

Normalizing as in the previous equations, the target document's model becomes:

$$\mathcal{D}_t \approx \sum_{s=1}^{|\mathrm{S}|} \lambda_t^s \mathcal{F}_t'^s + \lambda_t^n \mathcal{F}_t'^n \tag{1.9}$$

These new factor tensors, which are no longer necessarily rank 1 tensors, contain the proportions of related n-grams, separated into components according to their attributed source. This comprises the sought after semantic model of the document.

Given these new factors, the influence of each document is extracted:

$$\Lambda_t = (\lambda_1, \lambda_2, \ldots \lambda_{|S|}, \lambda_t) \tag{1.10}$$

Weights for each document are then extracted as their proportion of importance to the target document.

$$\mathrm{W} = \frac{1}{\sum \Lambda_t} \Lambda_t \tag{1.11}$$

Note that the weights from the source documents are not used. Essentially, the only purpose the factors of the source documents serve is to classify the factors of the target document. Having accomplished the classification step and constructed the model in Equation 1.9, and extracted the weights in Equation 1.11, the target document has been decomposed into a set of tensors which identify both the semantic shape of each contribution and its corresponding weight.

1.1.4 Summary of Influence Modeling Procedure

Generating the above model can be subdivided into the following steps:

1. Encode each document in the corpus as a tensor.

2. Decompose each document using non-negative CPD.

3. Classify each factor of the target document $\mathcal{D}_t$ as either belonging to a source document or as an original contribution of the author.

4. Extract weights from each subset of factors to determine the influence of each class of factors.

The details of how each of these steps is accomplished appears in the next chapter of this dissertation.

1.2 Related Work

Much of the inspiration for frequency based analysis for authorship detection comes from the work of John Burrows and Hugh Craig [5, 6, 10]. In their papers, Burrows and Craig utilize a variety of numerical techniques to explore marker words, and they use frequencies of marker works coupled with T-distribution sampling to provide an argument for attributing authorship of disputed works. They explore a variety of literary works, ranging from poetry to Shakespeare's plays. (Most of their focus is on Elizabethan and Victorian era works.) In all of these works, the frequencies explored are based on single marker words, and the words are extracted based on how unique they are to the authors in question.

For n-gram classification, Noriaki's Kawamae's paper has shown that n-grams are capable of building a generative topic model of a corpus of documents [17]. Kawamae's work shows that a combination of n-gram and word frequencies reveals information about a corpus's structure, especially hierarchical information pertaining to topics within the corpus. Kawamae's model builds a tree with probabilistic relationships which are then used to infer information about the structure of a corpus, and shows that n-grams provide a sufficient basis for modeling the transfer of ideas through a corpus.

Another related n-gram study was performed by Antonia, Craig, and Elliott [2]. In this paper, Antonia et al. attempt to reproduce marker word studies using n-gram frequencies in place of word frequencies. They were able to show that n-gram frequencies are able to identify stylistic signatures of contributors to a text document. When $n = 1$, their model is equivalent to marker words, and as they increase n, they retest to determine how expressive the model is. They noted that there is no one length that seems to work best in all cases when analyzing English language documents. Their results show that 1-gram, 2-gram, and 3-gram analysis tends to work well, but when exploring longer phrases the power of the model drops off. Even in instances where 1-grams or 2-grams are best, 3-grams are still a reasonable choice. Based on this result, 3-grams will be used in the tests in this dissertation. The analysis performed in Antonia et al.'s work was conducted using delta and zeta tests as was established in the standard marker word approach. As such, only the most frequent n-grams of each author were explored, and they were only used as an evidentiary marker of an author's participation. One advantage that the analysis proposed in this dissertation has is that it will account for all n-grams, and will explore how n-grams relate to each other within the target document's structure.

1.3 Outline of this Dissertation

The rest of this dissertation is organized as follows. First, there is a chapter detailing the approach of building, fitting, and evaluating the influence model. Following the approach explanation is an application which analyzes a conference paper and its sources. The final chapter discusses the findings in the case study as well as some notes for further application of this analysis technique.

Chapter 2

Approach

This chapter details the approach used to build the influence model for a target document and its corpus of supporting documents. This chapter also covers details of implementing this model and the challenges inherent in realizing this model. The first section of this chapter outlines the various steps required to build the influence model. The chapter concludes with implementation challenges and details.

2.1 Influence Modeling

The influence model is governed by a document list and a set of parameters. The inputs to the model are described in Table 2.1. The document list contains a list of all of the documents in the corpus and is comprised of potential source documents and one target document. The target document is placed at the end of the document list by convention.

The generated model's output consists of the set of factors which have been found to influence the target document, the weights of each document's influence on the target document, and the set of factors found from the decomposition of the document tensors. The output variables of the generated model are described in Table 2.2.

Table 2.1: Model Input

Parameter	Explanation
$docs$	A list of documents in the corpus. The target document is the final entry in the list.
n	The number of modes to use in tensor construction.
$nfactors$	The number of factors for tensor decomposition.
$threshold$	The threshold value for factor matching.

Table 2.2: Model Output

Parameter	Explanation
W	Set of weights of each factor of the target document. W_i is the weight of target document factor i.
S	The set of source indexes for each factor. S_i is the index of the source factor, 0 if the factor is unique to the target document.
F	The set of all document factor tensors.

2.1.1 Approach Overview

The overall process was described in Chapter 1. What remains is to see the detailed formulation of how each component of the model is computed. The overall algorithm is described in Algorithm 1. The principal activities in the model building process are document preparation, tensor construction, and influence extraction.

2.1.2 Document Filtering and Vocabulary Extraction

The first step is to prepare the document corpus as detailed in Algorithm 2.The documents are left mostly intact with the only filtering being to remove punctuation, numbers, and convert to lower case. The document strings are then treated as a list of lower case words and will be treated as such for the rest of this explanation.

Note that none of the words, including stop words, of the document are removed during filtering, and no stemming is performed. The reason for this is that these elements go to the style of the author. In fact, both stop words and unstemmed words have been shown to be powerful markers for authorship and style [2, 23, 6].

In order to build tensors, a vocabulary is first extracted from the corpus. The vocabulary is simply the set of all words within the corpus. The set V contains a single entry for each word. The index of each word is used in the next step to create tensor representation of each document. This process is described in Algorithm 3.

2.1.3 Tensor Construction

Following the preparation of the corpus and vocabulary extraction, the next step is key to the construction of the tensor model. In this step, each document is represented by a tensor. The tensor is constructed with n modes where each mode contains $|\mathrm{V}|$ dimensions. For example, 3-mode tensor over a a 30-word vocabulary would result in a $30 \times 30 \times 30$ tensor. This is used to count the frequency of n-grams within each document. Thus, entry $\mathcal{D}_{ijk}$ counts the number of occurrences of the phrase $\mathrm{V}_i\mathrm{V}_j\mathrm{V}_k$ within the document.

```
input  : $docs$, $n$, $nfactors$, $threshold$
output: W, S, F
prepare($docs$);
V $\leftarrow$ build_vocabulary($docs$);
C $\leftarrow \emptyset$;
foreach $d$ in $docs$ do
    $\mathcal{D} \leftarrow$ build_tensor ($d$, $n$, V);
    C $\leftarrow$ C $\cup \{\mathcal{D}\}$;
end
$\Lambda$,F $\leftarrow$ extract_factors(C, $nfactors$);
$M \leftarrow$ build_distance_matrix(F);
$\lambda \leftarrow$ the entries in $\Lambda$ corresponding to the target document.;
W, S $\leftarrow$ extract_influence($|docs|$, $M$,F,$\lambda$, $threshold$);
return W, S, F;
```

Algorithm 1: Influence Model Construction

```
input  : $docs$
output: None
foreach $d$ in $docs$ do
    Remove Punctuation from $d$;
    Remove Numbers from $d$;
    Convert $d$ to lower case;
end
```

Algorithm 2: Prepare

```
input  : $docs$
output: V
V $\leftarrow \emptyset$;
foreach $d$ in $docs$ do
    foreach $word$ in $d$ do
        V $\leftarrow$ V $\cup \{word\}$;
    end
end
return V;
```

Algorithm 3: Build Vocabulary

The construction of this tensor is detailed in Algorithm 4. The tensor is constructed using a sliding window, beginning with the first word of the document and proceeding until n words from the end of the document. This process is illustrated in Figure 2.1.

The resultant tensor is typically very sparse as it counts the frequency of all possible n-grams over the vocabulary V, and intuitively few (if any) documents would use every possible n length combination of its vocabulary as most of these phrases would be nonsensical. Thus the set of tensors C produced by Algorithm 4 is a set of sparse tensors representing the document corpus. The last tensor in the set is the representation of the target document as it has been placed at the end of the document list, as mentioned previously.

```
input  : $d$, $n$, V, $n$
output: $\mathcal{D}$
$\mathcal{D} \leftarrow$ Tensor with dimension $|V| \times |V| \ldots \times_n |V|$;
Fill $\mathcal{D}$ with 0;
$len \leftarrow$ number of words in $d$;
for $i \leftarrow 1$ to $len - n$ do
    /* Compute Tensor Element Index                    */
    $index \leftarrow$ list of $n$ integers;
    for $j \leftarrow 1$ to $n$ do
        $index[j] \leftarrow$ index of word $d[i]$ in V;
    end
    /* Update Frequency of This $n$-gram               */
    $\mathcal{D}[index] \leftarrow \mathcal{D}[index] + 1$;
end
return $\mathcal{D}$
```

Algorithm 4: Build Tensor

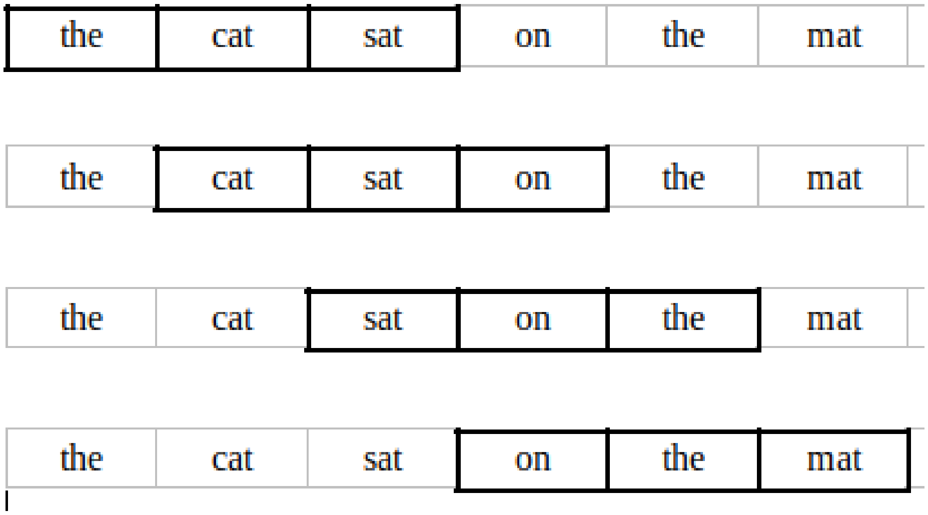


Figure 2.1: Sliding Window

2.1.4 Tensor Decomposition

The next step in the process is to decompose each document tensor into rank-1 components. As noted in Chapter 1, several decompositions exist which can accomplish this task. Because the data the present model consumes is frequency counting, the tensors comprising the corpus are all strictly non-negative. This arrangement naturally lends itself to non-negative factorization. Moreover, as these tensors are expected to be very sparse, any factorization that admits negative numbers would likely take a very long time to converge as it explores many combinations of factors which sum to zero. In fact, these alternating negative and positive factors would necessarily dominate the factors and would make comparison of factors very difficult while providing no useful information about the underlying document. Therefore, not only is non-negative factorization a logical choice, it is also a necessary choice to ensure the expressiveness of the resultant model.

The method of non-negative factorization employed in this model is the Columnwise Coordinate Descent (CCD) method described in Ji Liu et al.'s paper [21]. The CCD method decomposes a tensor $\mathcal{A}$ into a core tensor $\mathcal{C}$ and a set of factor matrices $U_{1 \dots m}$ where m is the number of modes. The result of this decomposition is shown in Equation 2.1. The object of the model is to minimize the error tensor $\mathcal{E}$, and CCD accomplishes this by iteratively solving for U_i by holding the other factor matrices constant. The optimal solution for each entry of U_i is determined by a differential equation which is solved iteratively as it has no closed form solution. The big advantage to the CCD method is that rows in each column are independent, and so entire columns can be solved in parallel. CCD also allows for L_1 sparsity constraints to be applied, though this is not used in the present model. (The L_1 penalty is set to 0 for this model.)

$$\mathcal{A} = (\mathcal{C} \times_1 U_1 \times_2 \dots \times_m U_m) - \mathcal{E} \tag{2.1}$$

Note that the CCD model is a non-negative version of the Tucker decomposition. By constraining the CCD model to use a square identity tensor (of dimension $n \times n \times \dots n$) for $\mathcal{C}$, the model becomes equivalent to the non-negative canonical polyadic decomposition. Each U_i matrix will contain n columns, and when this product is carried out, it can be rewritten as the sum of the tensor product of the columns of U. That is, the equivalent CP model can be expressed using Equation 2.2.

$$\mathcal{A} = \sum_{i=1}^{n} U^1_{:,i} \otimes U^2_{:,i} \otimes \ldots \otimes U^m_{:,i} - \mathcal{E} \tag{2.2}$$

Having extracted the rank-1 tensors which approximate $\mathcal{A}$, the last remaining step is to normalize these factors. The norm used in this model is the L_1 norm. Separating these out, the final approximation of each document tensor is shown in Equation 2.3.

$$\mathcal{A} \approx \sum_{i=1}^{r} \lambda_i \mathcal{F}_i \tag{2.3}$$

The L_1 norm is used here, and in the distance calculation in a later step, because it produces a sparse solution. The tensors being factored in this model are already very sparse, and the non-negative decomposition produces factors which are also sparse. Using the L_2 norm would tend to allow large differences between the factors to dominate the model's selection of related factors when the differences between factors are computed. Because the lower sensitivity of the L_1 norm is desired in the distance calculation, it is also used here. Doing so sets the range of distances between factors to the interval $[0, 2]$.

The entire process of the construction of these factors is shown in Algorithm 5. The result is an L_1-normalized set of rank-1 tensors.

As noted in Chapter 1, the number of factors determines the uniqueness of the decomposition. In the case of canonical polyadic decomposition, the solution is unique if the number of factors exceeds the tensor rank. However, computing the tensor rank is intractable, and so it must be approximated through trial and error. One rule of thumb for a 3-mode tensor, which is what is used in the case study in this dissertation, is that its expected minimal rank is given in Equation 2.4 [9].

$$R = \left\lceil \frac{IJK}{I + J + K - 2} \right\rceil \tag{2.4}$$

However, this estimate assumes a generic tensor and not a sparse tensor! In fact, in every instance of the document tensors used here, this minimal rank would far exceed the number

```
input  : C, $nfactors$
output: $\Lambda$, F

F ← $\emptyset$;
$\Lambda$ ← $\emptyset$;
$nmodes$ ← number of modes in C[1];
foreach $\mathcal{D}$ in C do
    U ← ccd_ntfd($\mathcal{D}$, $nfactors$);
    for $i$ = 1 to $nfactors$ do
        /* Build the Factor                                        */
        $\mathcal{T}$ ← U[1][:, $i$];
        for $m$ = 2 to $nmodes$ do
            $\mathcal{T} \leftarrow \mathcal{T} \otimes$ U[$m$][:, $i$];
        end
        /* Compute the norm and normalize the factor               */
        $\lambda$ ← $L_1$_norm($\mathcal{T}$);
        $\mathcal{T} \leftarrow \mathcal{T}/\lambda$;
        /* Insert the factor and norm into the list                */
        F ← F $\cup \{\mathcal{T}\}$;
        $\Lambda \leftarrow \Lambda \cup \{\lambda\}$;
    end
end
return $\Lambda$, F
```

Algorithm 5: Extract Factors

of non-zero elements. This leaves the choice up to searching for a number of factors which gives the best fit to the model. The approach used to find this number begins with assuming that the non-zero elements, nnz, of the document tensor were packed into a dense tensor with dimensions $\sqrt[3]{nnz} \times \sqrt[3]{nnz} \times \sqrt[3]{nnz}$. This starting point is then computed using Equation 2.5. From this point, decompositions are attempted with increasing rank until the fit begins to become worse, or until the error ratio drops below 20% (Equation 2.6). (Of course a different threshold could be used if desired.)

$$R_0 = \left\lceil \frac{nnz}{3\sqrt[3]{nnz} - 2} \right\rceil \tag{2.5}$$

$$\frac{|\mathcal{D} - \hat{\mathcal{D}}|}{|\mathcal{D}|} \tag{2.6}$$

Extending the starting point from Equation 2.5 for tensors of arbitrary, n, modes yields Equation 2.7. However, the rank of sparse tensors is very much an open question. There is no real theoretical basis for these equations other than sensible conjectures. The process of finding the number of modes is, for now, confined to a process of trial and error. The objective is always to find a model that fits reasonably well, and until the problem of sparse tensor rank is solved this is all that can be achieved without an exhaustive search of all possible ranks. These starting points do seem to yield good results in practice, and as 3-grams have been shown to work best for author classification [2], 3-mode tensors (and Equation 2.5 will be used in all of the experiments in this dissertation.

$$R_0 = \left\lceil \frac{nnz}{n\sqrt[n]{nnz} - 2} \right\rceil \tag{2.7}$$

2.1.5 Factor Classification

Having extracted factors from the document corpus, the next step is to classify each of the target document's factors as either belonging to the set F_t^s (target factors with sources) or F_t^n (target factors without sources). In order to do this, the similarity of each factor pair must be measured. Because each factor has the same dimension, and each factor's modes represent indexes over the same vocabulary, they can be compared by distance from each other within the factor space. Algorithm 6 accomplishes this task by finding the L_1 distance between each pair of factors. The result is a matrix M where M_{ij} is the L_1 distance between F_i and F_j. Of course, this matrix will have zeroes on the diagonal. Because each factor is non-negative and already L_1 normalized, $0 \leq M_{ij} \leq 2$, where 0 is a perfect match and 2 indicates maximum distance.

In actuality, only the entries corresponding to the target document factors are necessary. That is, $M[i, j]$ is only needed where i is the index of a target factor and j is the index of a potential source factor. The entire distance matrix is useful for studying the distribution of factor distances which is useful in finding model thresholds as well as quantifying the uniqueness of each factor.

The final task to be performed in constructing the model is to identify which source factors are closest to each target factor and compute the corresponding weights of those factors. This task is carried out by Algorithm 7. The basic strategy is for each factor in the target document to be assigned the source factor with the minimum distance. The only issue with this approach is it would always assign a source to a target factor, even though some target factors are expected to have no relatable source. For this reason, two steps are needed. First, the minimum is found, second it is compared against a threshold. If the minimum value is below this threshold, the factor is assigned a source. If, on the other hand, the minimum distance is above the threshold it is not assigned a source.

The threshold value is a heuristic parameter which controls the matching of factors. Recall that the factors are L_1-normalized, and the distance computed between the factors is the L_1-distance. If two factors are a perfect match, this results in a distance of 0 If they are completely disparate, the result will be a maximum distance of 2. This latter arrangement implies that no non-zero entries in the factor tensors were found in the same position and would therefore signify completely unrelated factors. A sensible default setting for this threshold is 0.2 as this requires a 90% agreement of the entries. Another approach to selecting a threshold value is to examine the distribution of distances within the distance matrix and use that information to select the threshold. For the applications in this dissertation, a threshold value of 0.2 is used.

```
input  : F
output: $M$

$M \leftarrow$ Matrix with dimension $|F| \times |F|$;
for $i = 1$ to $|F|$ do
    for $j = 1$ to $|F|$ do
        $M[i,j] \leftarrow$ L_1_norm(F[$i$] − F[$j$]);
    end
end
return $M$
```

Algorithm 6: Build Distance Matrix

```
input  : $ndocs$, $M$, F, $\lambda$, $threshold$
output: W, S

/* Compute Weights                                  */
$sum \leftarrow \sum \lambda$;
W $\leftarrow \lambda / sum$;
S $\leftarrow$ list of integers of size $|\lambda|$;
/* Classify Factors                                 */
$nfactors \leftarrow |\lambda|$;
for $i = 1$ to $nfactors$ do
    $min \leftarrow M[row, 1]$;
    $minIndex \leftarrow 1$;
    $row \leftarrow i + nfactors * (ndocs - 1)$;
    for $j = 1$ to $nfactors * ndocs$ do
        if $M$ [row,j]< $min$ then
            $min \leftarrow M[row, j]$;
            $minIndex \leftarrow j$;
        end
    end
    if $min \leq threshold$ then
        S[$i$] $\leftarrow minIndex$;
    else
        S[$i$] $\leftarrow 0$;
    end
end
return W, S;
```

Algorithm 7: Extract Influence

The result of the classification operation is the set W which is simply the normalized set of λ values for the target document, and the set S where entry S_i is the index of the factor which is the source for target factor i. If target factor i has no assignable source, then a value of 0 is written to position S_i.

After the factors have been matched, the final output of the model can be summarized by summing the influence of each source document and author contribution factor using Algorithm 8.

```
input  : $ndocs$, S, W
output: I, $author$

I $\leftarrow$ List of 0 repeated $ndocs - 1$ times;
for $i$ = $1$ to $ndocs$ do
    if S[$i$] $= 0$ then
        $author = author +$ W[$i$];
    else
        $j \leftarrow$ Document number corresponding with S[$i$];
        I[$j$] $\leftarrow$ I[$j$] $+$ W[$i$];
    end
end
```

Algorithm 8: Final Summation

2.2 Implementation

Implementing the model described in this chapter comes with several challenges. The biggest challenge is the size of the tensors, as well as a lack of good support for sparse tensors in available software. Several packages were tried, but ultimately a custom tensor library was needed to support these tensors. The attempted software packages were Tensor Flow (Python), Tensor Toolbox (Matlab), SciPy/NumPy (Python). While all three packages provide support for sparse tensors, their operations are not well optimized for sparse tensor usage. Also, in some cases, tensors were converted into dense tensors before operations were performed. Unconstrained vocabularies can often have tens of thousands of words, which leads to a tensor which far exceeds the capacity of any machines available for this project. Before these libraries were abandoned, constrained vocabularies were attempted.

2.2.1 Constraining Vocabularies

As already stated, the tensors used in this model are very sparse. Essentially, every word in a document is the beginning of a new phrase, and so every document tensor will contain the same number of non-zero entries as there are words in the document. (With the exception being the last n-gram as each word following the first in the n-gram cannot be the start of a new n-gram.) Storing the frequency counts of these documents is trivial, but the model needs them in their positions within the tensor in order to decompose and fit elements. Most of the complexity, therefore lies with the dimensionality of the tensor which is driven by the size of the vocabulary.

Looking at the vocabularies in several documents during preliminary experiments showed that each document only had about five hundred to one thousand frequent words. By sorting the vocabulary in descending order by frequency, the vocabulary can be shortened with minimal disturbance to the structure of the document and the makeup of most of the n-grams. Constraining the vocabulary to 600 words when building a 3-gram tensor results in a tensor which has 216,000,000 potential entries. Even in dense format, this can be comfortably accommodated by the memory of even a modest modern desktop machine. However, a problem still remains with this approach. Decompositions of a tensor of this size, typically into 100-200 factors, requires too much time. When using Matlab on an 8-core 3.4GHz Intel Linux machine with 16GB of RAM, non-negative factorization tended to require about 2 hours of elapsed wall-clock time, and so a faster method was still needed even in the constrained case. Searching for the optimal number of factors by multiple factorings proved even more challenging. This is especially challenging when considering that this model requires the decomposition of multiple documents within a corpus.

2.2.2 The sptensor Library and Tool

To address the problems of time and memory constraints, a new library was created. This library is called sptensor, and it is written in ANSI C. C was selected because it provides enough control for optimizing memory usage and it is well situated to have libraries for other languages bound to it.

The sptensor library is implemented as a shared library, and also has a command line tool which allows tensor operations to be performed on files. The main components of the sptensor library are:

`vector` An array list styled general purpose storage structure. This grows dynamically as needed.

`sptensor` A sparse tensor storage structure. Tensors are stored as a list of coordinates with indexes and values stored in separate vectors. The tensor indexes are maintained in sorted order giving $O(\lg n)$ lookups.

`tensor_view` A series of overlays for tensor objects. These provide general ways of accessing tensors, printing tensors, and performing operations. Operations included in the library are reshaping tensors, slicing tensors, and general tensor arithmetic. Mechanisms are provided to allow the library's user to provide their own tensor views.

`tensor_math` A set of tensor operations. These include element wise operations, scalar multiplication, and a set of tensor products.

`ccd` An implementation of Liu's Columnar Coordinate Descent non-negative tensor factorization algorithm [21].

The goals for the development of this library are:

1. Implement a library for dealing with sparse tensors efficiently.

2. Provide fast CCD tensor decomposition.

3. Provide an MPI implementation for common tensor functions to allow for operations on very large tensors.

At the time of this writing, the first two goal have been achieved but unfortunately the MPI version of sptensor does not yet exist. The sptensor library is designed to only represent sparse tensors, and so it has the opposite problem the common tensor libraries have. Dense tensors in sptensor are therefore fairly inefficient both in time and space complexity. However, for the present application the library performs quite well. Where Matlab factorization of tensors over constrained vocabularies typically require 1-2 hours to complete on a modest desktop computer (3.4GHz), sptensor can accomplish the task in 5-10 minutes. This speedup allowed for the construction and testing of the model to proceed.

2.2.3 Text Modeling Suite

The text model described in this chapter is implemented as a series of stand-alone programs. Essentially, each of the algorithms described in this chapter are implemented as either part of the sptensor library or as a stand alone utility. The decision of whether to build an algorithm into sptensor was based on whether the operation was a generic tensor operation, or whether it was specific to this model. The sptensor tool provides the following facilities:

- CCD Factorization (as used in Algorithm 5)

- Extraction of normalized factors from the matrix output of CCD, also from Algorithm 5.

- Distance matrix computation (Algorithm 6)

In addition to the sptensor utility, the following standalone programs are used to build the model:

prepare This is a simple bash shell script which uses the UNIX command tr to perform the filtering and lower case conversion found in Algorithm 6.

vocabulary A small python program that reads all the documents in a corpus, builds a vocabulary, and counts word frequencies. The vocabulary is then sorted and truncated to a desired length. An @ symbol is inserted at the end as a wildcard for all the infrequent words. The output of this program is a vocabulary file, which simply lists each word in the vocabulary one per line.

doctns A C program which uses the sptensor library to build the document tensors according to Algorithm 4. If doctns encounters a word not in the vocabulary, it uses the wildcard at the end of the list.

classify A C program which uses the sptensor library to classify and report classification of factors. The output of this corresponds to the S and W sets in Algorithm 1.

build-model A shell script which invokes all the other programs as per Algorithm 1.

Chapter 3

Results

This chapter contains the results of two sample runs of the model. The first is a simple example which is intended to show what the model's results look like. The second example is the result of the classification of a regional conference paper.

3.1 A Simple Example

One of the problems with a model such as this one is that ground truth is difficult to find. In fact, this model quantifies a property which even the author of the documents in question may be unaware of. As such, verification of the model depends on the sensibility of its answers. However, this again presents a challenge because the total structure of all of the factors and relationships among them is very complex and difficult to visualize. To help alleviate this problem, this section contains a very simple example, one in which all of the components of the model may be seen. This example uses three simple stories, written for this purpose, drawing on a 30 word vocabulary. The first story is in Figure 3.1, the second is in Figure 3.2, and the third is in Figure 3.3. The third story is intended to be a sequel of sorts, drawing on material from the previous stories. The complete vocabulary for this example is shown in Table 3.1.

The cat sat on the mat. The cat was happy to be on the mat. The cat saw the mouse running but was too lazy to chase it.

Figure 3.1: The Cat's Tale

The dog walked to the house. The dog saw the food bowl, and the dog saw a squirrel. The dog chased the squirrel from the food bowl.

Figure 3.2: The Dog's Tale

The dog saw the cat on the mat. The dog walked to the house, and the dog chased the cat. The squirrel was happy to see the dog chase the cat on the mat. The dog saw the squirrel, and decided to chase the squirrel instead. The cat sat on the mat.

Figure 3.3: The Saga Continues

Table 3.1: Cat and Dog Vocabulary

I	Word	*I*	Word
1	the	16	chased
2	house	17	sat
3	mouse	18	be
4	squirrel	19	happy
5	it	20	on
6	saw	21	from
7	lazy	22	food
8	cat	23	decided
9	mat	24	to
10	a	25	was
11	bowl	26	dog
12	walked	27	running
13	too	28	instead
14	and	29	but
15	see	30	chase

The number of factors was determined by trial decomposition of the target document. The target document consists of 52 words, which yields 49 3-grams, of which 42 are unique. The resultant document tensor has 42 non-zero entries. Of course, one unique solution to the decomposition of this tensor would be 42 tensors consisting of the individual non-zero entries in the same position as in the target tensor. This would be the maximal candidate for the rank of the tensor. Tensor decompositions were performed for between 1 and 42 factors. Above 15 factors, the CCD algorithm produced factorizations that were mostly zeroes. That is, most of factor tensors consisted entirely of zero entries. This is due to the small number of entries present, and the small amount of variation between them. Because the higher-ranked factorizations all suffered from this problem, they were excluded from consideration. For the factorizations between 1 and 15 factors, 7 yielded the best fit with an error ratio of 0.45, and so 7 was determined to be the optimal number of factors for this corpus. Even with this relatively poor fit, the model was able to distinguish features of the stories. The distances between the factors, other than the diagonals, range from 0 to 1.4. The final classification step yielded the classification of factors shown in Table 3.2.

The model threshold value was set to 0.2, which is a sensible value as this requires a 90% match in a factor. In this case it would not have mattered, however, as the matching factors were all perfect matches with an L_1 distance of 0.

Factor two was matched to the cat story, and factor seven was matched to the dog story. These factors are shown in Table 3.3 and Table 3.4. As can be seen in these factors, the principal takeaway from the cat's story is that it was "on the mat", a theme which is indeed carried to the target document. For the dog's contribution, the 3-grams show the various actions that the dog performs, and in the sequel the dog is the clear actor (for better or worse). The remaining factors (1,3,4,5, and 6) were determined to be original to the target document. These factors are seen in Table 3.5. The unmatched factor includes interactions which were not in the original stories, as well as actions the animals did not perform in the original.

The separation that the model accomplished, even with very scant data, shows that the expected similarities in the resultant factors are present. The next step is to look at a more substantial example.

Table 3.2: Cat Dog Model

Factor	Factor Weight	Classification
1	0.28	Author Contribution
2	0.15	Cat Factor 1
3	0.14	Author Contribution
4	0.14	Author Contribution
5	0.11	Author Contribution
6	0.11	Author Contribution
7	0.06	Dog Factor 1

Table 3.3: Factor 2 - Matched to Cat Factor 1

Word 1	Word 2	Word 3	Proportion
on	the	mat	1.00

Table 3.4: Factor 7 - Matched to Dog Factor 1

Word 1	Word 2	Word 3	Proportion
the	dog	saw	0.40
the	dog	walked	0.20
the	dog	chased	0.20
the	dog	chase	0.20

Table 3.5: Sequel Original Factors

Word 1	Word 2	Word 3	Proportion
saw	the	squirrel	0.267417
saw	the	cat	0.223651
saw	the	dog	0.192194
cat	the	squirrel	0.044066
cat	the	cat	0.036854
cat	the	dog	0.031670
mat	the	squirrel	0.034331
mat	the	cat	0.028712
mat	the	dog	0.024674
see	the	squirrel	0.032132
see	the	cat	0.026873
see	the	dog	0.023094
chased	the	squirrel	0.013437
chased	the	cat	0.011238
chased	the	dog	0.009657
squirrel	and	happy	0.249836
squirrel	and	decided	0.262960
squirrel	was	happy	0.237368
squirrel	was	decided	0.249836
decided	to	chase	1.000000
happy	to	see	1.000000
cat	saw	the	0.345830
cat	see	the	0.040819
cat	chased	the	0.172914
cat	chase	the	0.213734
walked	saw	the	0.056987
walked	see	the	0.006726
walked	chased	the	0.028493
walked	chase	the	0.035220
to	saw	the	0.044398
to	see	the	0.005240
to	chased	the	0.022199
to	chase	the	0.027439

3.2 A Conference Paper Case Study

For a real-world case study, a conference paper was pulled from the ACM Digital library. This was the first paper listed in the first conference listed in their regional conference proceedings. This paper cites four other papers and two websites. The two websites were used to pull data, and so they are not included in the corpus. In addition to the four cited papers, two unrelated papers are included to test if the model will select factors from these unrelated papers. The complete corpus, listed with the target paper last, is shown in Table 3.6. Documents 1-4 are the papers cited by the target paper, documents 5-6 are unrelated papers, and document 7 is the target paper.

The entire corpus consists of 45,152 words. As described in the previous chapter, the vocabulary was truncated to 600 words. The 600 words were the most frequent words across the corpus. The other parameters are shown in Table 3.7 Again, 0.2 is used as the threshold as it is a good default setting. The decomposition of the 7 documents into 150 factors was carried out on a machine with a 3.9GHz 8-core Intel processor and 15GB of RAM. The decomposition and construction of normalized tensors took approximately 2.5 hours to complete. Calculation of the distance matrix and classifying the factors required another hour and a half. The results are shown in Table 3.8.

The distribution of the factor distances are shown in Figure 3.4. The vertical red line on the graphs shows the threshold for factor matching. Figure 3.4(a) shows the distribution of factor distances for the entire factor matrix while 3.4(b) shows the distribution of the distances from the target factors. Note that both distributions are tri-modal. The two spikes on the far right correspond to factors from the unrelated documents, which shows that they are well separated from the target document's factors and from each other.

Table 3.6: Conference Paper Corpus

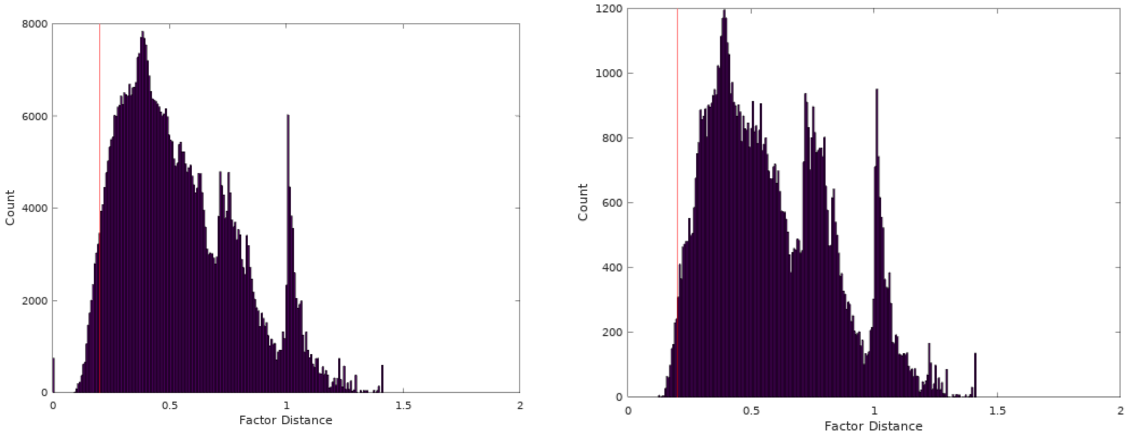
Num	Document Information
1	Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. A symbolic representation of time series, with implications for streaming algorithms. In Proc. DMKD 2003, pages 211. ACM Press, 2003.
2	Andreas Schlapbach and Horst Bunke. Using hmm based recognizers for writer identification and verification. In Proc. FHR 2004, pages 167172. IEEE, 2004.
3	Yusuke Manabe and Basabi Chakraborty. Identity detection from on-line handwriting time series. In Proc. SMCia 2008, pages 365370. IEEE, 2008.
4	Sami Gazzah and Najoua Ben Amara. Arabic handwriting texture analysis for writer identification using the dwt-lifting scheme. In Proc. ICDAR 2007, pages 11331137. IEEE, 2007.
5	Kolda, Tamara Gibson. Multilinear operators for higher-order decompositions. 2006
6	Blei, David M and Ng, Andrew Y and Jordan, Michael I. Latent dirichlet allocation. 2007
7	Serfas, Doug. Dynamic Biometric Recognition of Handwritten Digits Using Symbolic Aggregate Approximation. Proceedings of the ACM Southeast Conference 2017

Table 3.7: Conference Model Parameters

n	$nfactors$	$threshold$
3	150	0.2

Table 3.8: Conference Classification Results

Document	Influence	Number of Matched Factors
1	0.21	10
2	0.09	9
3	0.06	3
4	0.06	1
5	0.00	0
6	0.00	0
Author	0.57	127



(a) All Factor Distances

(b) Target Factor Distances

Figure 3.4: Factor Distance Distribution

Given the separation of the factors, and that the model excluded two obviously unrelated papers, the model appears to have produced the desired results. For final verification, the text of the documents in question will now be summarized. The target paper details an algorithm which is used to identify handwritten characters [22]. The algorithm presented is an extension of another algorithm, Symbolic Aggregate Approximation (SAX), which was first presented in Lin et al.'s paper [20]. Because the target paper is an extension of this one, a reasonable assumption would be that it would be heavily influenced by this source. This is precisely what the model stated in that this paper was assigned a weight of 0.21, and had a total of ten matching factors. The remaining citations that were found to influence the document present competing algorithms, and are primarily mentioned in the target paper's related works section. A summary of one of the most influential of the 10 matched factors is shown in Table 3.9. This is the factor 56 (out of 150) in the decomposition of the target document and represents a weight of 0.04. The factor was sorted in decreasing order by proportion and only the first 30 n-grams are shown. (The actual factor contains 11,661 n-grams, most of which have very small proportional entries.) Note most of the n-grams are discussing the SAX algorithm, and various properties of it. In fact, this is what is found in the other nine factors, they all discuss different elements of SAX.

Another property of the target paper that bears examination is the makeup of the text itself. The paper is 4 pages long, the first page being devoted to front matter and the related works. The second page contains the conclusion of the related works section, which occupies approximately 25% of the page. The rest of the second page, and the entirety of the third page have the author's contributions and the conclusion. Half of the fourth page has the final conclusion paragraph, and then the bibliography. By rough estimate, therefore, the paper contains 1.75 pages of what is essentially the summary of existing work. This leaves 2.25 pages of original material, which means that a cursory analysis of the paper would imply that the author has contributed 56% of the text of the paper. The model's output weight for the author's contributions of 57% is in line with this rough estimate.

As these results show, the model makes a set of reasonable matches, and it does not select unrelated documents. The actual makeup of the factors are much more complex, however ideas can be traced through them. Unfortunately, these factors are much too large to be included verbatim in this chapter. However, the factors that were matched from paper one all deal with a technique which generates a symbolic representation of a time series. This technique serves as the basis for the invention in the paper, and is talked about many times with many of the same explanations used in the first paper. Thus the model has not only avoided unrelated information, it has given a greater weight to the paper which had the greatest semantic influence on the work being studied.

Table 3.9: First 30 Non-Zero Entries of Factor 56

Word 1	Word 2	Word 3	Proportion
is	the	sax	0.000887
into	the	data	0.000886
symbols	the	sax	0.000874
digits	the	timeseries	0.000865
digits	the	data	0.000857
with	the	sax	0.000856
from	the	square	0.000852
however	the	sax	0.000844
up	the	sax	0.000844
characters	the	sax	0.000844
becomes	the	sax	0.000844
note	the	sax	0.000843
using	the	sax	0.000841
for	the	square	0.000838
into	the	sax	0.000838
from	the	svc	0.000833
from	the	paa	0.000832
author	the	square	0.000828
on	the	author	0.000824
for	the	svc	0.000819
for	the	paa	0.000818
on	the	accuracy	0.000814
on	the	array	0.000814
digits	the	sax	0.00081
author	the	svc	0.000809
author	the	paa	0.000808
on	the	distance	0.000806
on	the	x	0.000806
from	the	timeseries	0.000804
of	the	author	0.000802

Chapter 4

Conclusions

This chapter contains a summary of the results, a justification of the model, an outline of various weaknesses of the model, and then concludes with a discussion of future work.

4.1 Model Performance

As can be seen from the experiments in the previous chapter, the model performs as expected. The factors discovered by non-negative tensor decomposition all contain related n-grams. Moreover, the factors are unique enough that a match, or a near match within some heuristic bound, provides evidence of a relationship between factors. The related factors also make sense on an intuitive level, each having a clear semantic relationship to both target and source material.

The quantifications of the influencing factors also perform as expected. While no ground truth is available for a weighted mixture of source documents to the target documents, inspecting the documents in the corpus shows that the model's output reasonably matches the expectations of a human reader.

4.2 Justification of the Model

This influence model is based on a factorization of tensors describing the n-gram frequency counts of the document. n-grams are a fairly common approach to modeling the topic and style of documents, and therefore the topic and styles of the documents can logically be said to be contained within the tensor representation of them. By decomposing a document tensor into a non-negative polyadic decomposition, the resultant factors will be a mixture of covariant and contravariant factors. That is, the frequencies of the n-grams found within each factor will have similar covariant properties. This is done irrespective of the order of the n-grams within the document.

The next step of the model normalizes all the factors. In so doing, this removes the magnitude of the frequencies and leaves the factor tensors with a proportional profile of the n-gram composition of the document. This factor now contains a description of each of the principal elements of the document in relationship to its vocabulary.

By repeating this process for every document in the corpus, this technique produces a set of proportional profiles that describe the make up of each document within the corpus. By searching for commonality among these factors, the influence model locates documents which have common explanatory factors. If a document has a strong enough match on one or more of its factors, it provides evidence of a relationship between the documents. This evidence can be considered to vary in strength according to the selection heuristic, and because the factors are based on the distribution of phrases within the document, this provides a sound model of document influence.

Another aspect of this operation that makes this a useful model is that it compares all factors irrespective of their influence in their original source document. That is to say if a source puts forward some topic, which is subsequently modeled as a proportional factor, and that topic is relatively unimportant in its source this will have no impact on how much influence it may exert in a target document. In traditional influence models, which are based on word frequency, this sort of relationship will not be found. However, with the present model it will, and its weight will be based upon the total impact it has on the target document.

4.3 Weaknesses of the Model

Perhaps the greatest weakness of the present model is that no ground truth is available. This sort of quantification of influence is not readily available, but given the justification of the model the information it discovers can still be considered useful.

Another problem with this model is that it is sensitive to three input parameters. n for the number of words, $nfactors$ for the number of decomposition factors and $threshold$ for the selection criteria. Of course, justifications for the selection of n and $threshold$ have already been discussed. In the case of n, the setting of 3 is a standard starting point in the field of n-gram analysis for English, but this would likely need to be tuned for other languages. The biggest impediment to successfully modeling influence in this fashion is the $nfactors$. In order to get a well-fit and unique document tensor decomposition, the rank of the tensors is needed. However, knowing the true value of this parameter is intractable and it must be searched for. Further study and research into the open question of the rank of sparse tensors would alleviate this problem.

Finally, this model is based on n-gram frequencies. As such, small documents are often difficult to model because they will have relatively few repeated n-grams. If the distribution of n-grams is completely uniform, this will also act as an impediment to meaningful tensor decompositions.

4.4 Future Research

The immediate future plans for this research involve the further development of the sptensor library. Further optimization is needed, as well as completion of its MPI interface. Following that, library bindings to higher level languages will be created.

Another application of the model is to replicate the studies conducted by Craig and Burrows [6, 10]. Addressing the problem of Shakespearean authorship using this model poses several unique challenges, not least of which is due to the inconsistencies in Elizabethan spelling (which necessitates the decomposition over full vocabularies). Additional applications will also be explored, including establishing chronologies of documents via topological sorting and modeling the influence flow through a hierarchical network of documents.

The effects of constrained vocabularies are another area which needs to be addressed. Following the authorship studies, another future effort will be to address the effect of the vocabulary size on the output of the model. Other aspects warranting further study are the effects of the various parameters of the model.

Finally, several experiments are under way to extend the reach of the model from influence modeling to plagiarism detection. This last branch will perhaps be the most important as it will examine not only plagiarism from one source document, but will take into account many potential documents of origin.

Bibliography

[1] Adler, R., Ewing, J., and Taylor, P. (2009). Citation statistics. *Statist. Sci.*, 24(1):1–14. 2

[2] Antonia, A., Craig, H., and Elliott, J. (2014). Language chunking, data sparseness, and the value of a long marker list: explorations with word n-grams and authorial attribution. *Literary and Linguistic Computing*, 29(2):147–163. 12, 16, 23

[3] Bader, B., W. Berry, M. W., and Browne, M. (2007). Discussion tracking in enron email using parafac. In Berry, M. W. and Castellanos, M., editors, *Survey of Text Mining*, chapter 8, pages 147–163. Springer. 3, 6

[4] Bro, R. (1997). Parafac. tutorial and applications. *Chemometrics and intelligent laboratory systems*, 38(2):149–171. 6, 8

[5] Burrows, J. (2006). All the way through: testing for authorship in different frequency strata. *Literary and Linguistic Computing*, 22(1):27–47. 11

[6] Burrows, J. and Craig, H. (2017). The joker in the pack?: Marlowe, kyd, and the co-authorship of henry vi, part 3. In Taylor, G. and Egan, G., editors, *The New Oxford Shakespeare Authorship Companion*, chapter 11, pages 194–217. Oxford University Press. 3, 11, 16, 45

[7] Cantuariensis, A. (c. 1078). *Proslogion*. Ordo Sancti Benedicti. 1

[8] Carroll, J. D. and Chang, J.-J. (1970). Analysis of individual differences in multidimensional scaling via an n-way generalization of “eckart-young” decomposition. *Psychometrika*, 35(3):283–319. 6

[9] Comon, P., Luciani, X., and De Almeida, A. L. (2009). Tensor decompositions, alternating least squares and other tales. *Journal of chemometrics*, 23(7-8):393–405. 21

[10] Craig, H. and Kinney, A. F. (2009). *Sheakespeare, Computers, and the Mystery of Authorship*. Cambridge University Press. 3, 11, 45

[11] Dietz, L., Bickel, S., and Scheffer, T. (2007). Unsupervised prediction of citation influences. In *Proceedings of the 24th international conference on Machine learning*, pages 233–240. ACM. 2

[12] Harshman, R. A. (1970). Foundations of the parafac procedure: Models and conditions for an" explanatory" multi-modal factor analysis. 6, 8

[13] Harshman, R. A. and Lundy, M. E. (1994). Parafac: Parallel factor analysis. *Computational Statistics & Data Analysis*, 18(1):39 – 72. 8

[14] Håstad, J. (1990). Tensor rank is np-complete. *Journal of Algorithms*, 11(4):644–654. 6

[15] Hitchcock, F. L. (1927). The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6(1-4):164–189. 4

[16] Jiang, Z., Liu, X., and Gao, L. (2014). Dynamic topic/citation influence modeling for chronological citation recommendation. In *Proceedings of the 5th International Workshop on Web-scale Knowledge Representation Retrieval & Reasoning*, pages 15–18. ACM. 2

[17] Kawamae, N. (2016). N-gram over context. In *Proceedings of the 25th International Conference on World Wide Web*, pages 1045–1055. International World Wide Web Conferences Steering Committee. 12

[18] Kiers, H. A. (1991). Hierarchical relations among three-way methods. *Psychometrika*, 56(3):449–470. 7, 8

[19] Kolda, T. G. and Bader, B. W. (2009). Tensor decompositions and applications. *SIAM Review*, 51(3):455–500. 4, 7, 8

[20] Lin, J., Keogh, E., Lonardi, S., and Chiu, B. (2003). A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 2–11. ACM. 40

[21] Liu, J., Liu, J., Wonka, P., and Ye, J. (2012). Sparse non-negative tensor factorization using columnwise coordinate descent. *Pattern Recognition*, 45(1):649–656. 8, 20, 29

[22] Serfass, D. (2017). Dynamic biometric recognition of handwritten digits using symbolic aggregate approximation. In *Proceedings of the SouthEast Conference*, pages 1–4. ACM. 40

[23] Stamatatos, E. (2011). Plagiarism detection based on structural information. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1221–1230. ACM. 16

Vita

Robert Lowe was born in Knoxville Tennessee on June 9, 1980. He had his first programming experience at the age of 8 on his family's Commodore Vic-20 computer. Throughout middle and high school he worked diligently to get as much computer time as he possibly could, continuing his BASIC and assembler programming. In high school he took up C and C++ programming, and was amazed to discover that people could actually make a career out of computer programming.

After graduating South Doyle High School in 1998, Robert attended Middle Tennessee State University for three years, and then left to work for Business Information Systems, a government contracting company. After working on several systems for government record keeping, Robert returned to MTSU in 2005 and finished his BS in Computer Science that same year. Upon graduation, Robert Joined JC Reed, a financial conglomerate, and also took a part-time teaching position at Draughon's Junior College in Murfreesboro. This latter position blossomed into a life long love of teaching and academia. after leaving JC Reed, Robert worked as an independent contractor for several years before entering graduate school at the University of Tennessee in 2008.

During graduate school, Robert supported his growing family by working as an independent software developer and adjunct instructor at Pellissippi State Community College, ITT Technical Institute, Knoxville College, and Maryville College. He also worked as a teaching assistant and research assistant at UT. These efforts resulted in a full time tenure track position at Maryville College, a position which Robert started in 2014.

Currently, Robert Lowe is the sole computer scientist on the full time faculty of Maryville College. Upon graduation, he plans to continue teaching at Maryville College.