On Statistical Sequencing of Document Collections

by

Ramya Thinniyam

A thesis submitted in conformity with the requirements for the degree of Doctor of Philosophy
Graduate Department of Statistical Sciences
University of Toronto

© Copyright 2014 by Ramya Thinniyam
This thesis was primarily motivated by the Documents of Early England Data Set (DEEDS), a collection of undated medieval property exchange charters written in Latin. The main goal is to devise automated statistical methods to estimate the temporal order in which a corpus of documents was written using only the words that appear in the corpus. Our interest lies in sequencing the documents, not dating them. The premise is that documents written closer together in time will naturally be more similar in content, and thus we propose the following two-step approach to sequencing: (1) obtain distance measures between pairs of documents (using only their word characteristics); (2) estimate the optimal ordering of the documents based on these distances.

We describe various types of distances that can be computed between pairs of documents. We then present three methods for sequencing a set of documents based on their pairwise distances. The first method sequences elements using a regression model on their pairwise distances and optimizes the corresponding Error Sum Of Squares (SSE) to estimate the ordering. The second method called the “Closest Peers” method minimizes the average distance of each document to its closest matching documents. The third is an MCMC approach that not only allows for sequencing, but parameter estimation and natural inferences about orderings on subsets of the documents. The performance of the sequencing methods are evaluated and compared via simulated data.

The methods that we describe in this thesis are not only applicable to the DEEDS corpus, but also for other collections such as: drafts/versions of works written by the same author, different documents written by the same author, different documents written on the same topic, etc. Application
of the sequencing methods are carried out on real document collections such as the DEEDS corpus and on various sets of drafts. As an addendum, we propose a distance-based method for classifying documents using a training set and apply it to the DEEDS corpus.
Acknowledgements

First and foremost, I would like to thank my supervisor, Andrey Feuerverger, for suggesting this interesting thesis problem. He has been an incredible teacher, mentor, and supervisor and an absolute pleasure to work with. Over the years, he has constantly supported, encouraged, and pushed me when needed. He has shared with me many ideas and insights that were invaluable and has maintained his enthusiasm for this work. He would question my ideas so that I could refine my thoughts and make them as clear as possible. I would like to convey my heartfelt gratitude to Andrey, without whom this thesis would not have been possible.

I am grateful to Professors Jeffrey Rosenthal and Michael Evans for serving as committee members. Their questions and comments during committee meetings helped me to improve the methods developed in this thesis. In particular, Jeff Rosenthal suggested the MCMC approach for sequencing and inference and offered other related suggestions to improve algorithms, such as Simulated Annealing. He was also generous in making time to clarify my questions about MCMC methods and was very helpful in the work of Chapter 5. Michael Evans was also very supportive and helpful in clarifying many of the issues arising in Bayesian viewpoints of the problem. I would also like to thank Professor Fang Yao and Michael Gervers for participating in my final oral defense. Special thanks to Professor Vladimir Vinogradov of Ohio University for coming to Toronto to serve as the external examiner. His thorough revision of the thesis as well as his comments and questions during the oral defense were valuable.

This thesis was originally motivated by the DEEDS project, which was founded by Michael Gervers, Professor of History, at the University of Toronto. I would like to thank him for generously providing me with the DEEDS data set to work with. Also, a big thank you to Roderick Mitchell, Oracle developer and instructor at the University of Toronto and for the DEEDS project. He assembled the DEEDS data sets for me and formatted and reformatted the data as required, and also provided me with valuable information needed for my analysis.

I am pleased to acknowledge that part of this research was financially supported by several grants
from Ontario Graduate Scholarship, and a grant from NSERC and Google Inc.

I thank my undergraduate professors, especially Alison Weir and Jerry Brunner, for sparking my interest in Statistics and for instilling in me a passion for the subject, as well as for advice and inspiration. My graduate professors also played an important role in teaching me various courses and building the vital skills required for research. I would like to especially thank the following professors for their teaching and advice during my graduate career: Don Fraser, Alison Gibbs, Radford Neal, Jeremy Quastel, Nancy Reid, Jeffrey Rosenthal, James Stafford, Bálint Virág, and Fang Yao.

A big thank you to all of the very friendly staff of the Statistical Sciences Department. Many thanks to Dermot Whelan for his frequent and timely help with server and computing issues and to Andrea Carter for her assistance with administrative issues and for her kind support throughout the years.

To my friends and colleagues Muz Mallo, Zeynep Baskurt, Tara Cai, Tara Kehoe, Avideh Sabeti, Shivon Sue Shee, Alex Shestopaloff, Becky Lin, and others: thank you for your moral support and friendship over the years. We have spent countless hours together discussing statistical and other issues and enjoying the company of one another. Each of you has been there to help with computing issues, clarifying statistical methods, and much more. Special thanks to Muz who helped give me programming ideas, debug code, and gave me the suggestion to analyze the versions of the Song of Solomon. I truly appreciate all of his help. I thank Gelila Tilahun whose previous work on dating DEEDS charters was very useful for me to reference. Her friendly advice and discussions on this topic were also very helpful.

Finally, I would like to thank my parents, Jaya and Balu; my sister, Priya and my brother-in-law, Arun; and my very cute newly born niece, Anisha, and also my extended family. They have always encouraged me in each of my endeavours and have allowed me to accomplish my goals. They have made me the person I am today. Thanks to my family for their constant love, support, and patience, especially during the stressful times.
Contents

1 Introduction 1

1.1 Text Processing Preliminaries ............................................. 2

1.2 On Orderings .................................................................. 3

1.3 Past Research ................................................................ 7

1.4 New Approach ................................................................ 15

1.5 Outline Of Thesis .............................................................. 16

2 Distance Measures 18

2.1 Types of Distance/Similarity Measures ................................. 19

2.2 Problems with using only Word Frequencies ......................... 24

2.3 Latent Semantic Analysis (LSA) ......................................... 25

  2.3.1 The Document-Term Matrix .......................................... 25

  2.3.2 Singular Value Decomposition and Rank Reduction ......... 26

  2.3.3 Obtaining Distances/Similarities using LSA .................... 28

  2.3.4 Using LSA for Information Retrieval .............................. 29

  2.3.5 Advantages and Limitations of LSA .............................. 30
2.4 Some Ideas Relating to Singular Value Decomposition

2.4.1 Searching for Distances that Align with Time

2.4.2 A Logistic SVD Model

2.5 The “Tableaux” Method

2.5.1 Tableaux-based Distances and Similarities

2.5.2 Searching for Optimal Values of $\alpha$, $\beta$, $\gamma$

2.6 Converting Similarities to Distances

3 Sequencing Elements via Least Squares Criteria

3.1 Sequencing Elements based on Distances

3.2 A Linear Regression Model for Distances

3.3 A Regression Model based on Spacings

3.4 Form of the Regression Matrices and Estimators

3.5 Properties of the Sum of Squared Errors (SSE)

3.6 Concerning Estimates of the Error Variance, $\sigma^2$

3.7 Constrained Estimation of Spacings

3.7.1 Estimation under Non-Negativity Constraints

3.8 Sequencing using NNLS vs. Unconstrained Regression

3.9 Generalizing the Criterion Function

3.10 A “Closest Peers” Method for Sequencing Elements

4 Optimization Algorithms for Sequencing Problems

4.1 “Greedy Take and Put One” Algorithm
4.2 Examples using “Greedy Take and Put One” Algorithm with SSE Criterion Function

4.3 Comparing the SSE and ADP Criterion Functions

4.4 Improvements to the “Greedy Take and Put One” Algorithm

4.5 Greedy “Fix T Take and Put One” Algorithm

5 Estimation and Inference on Sequencing via MCMC

5.1 Markov Chain Monte Carlo Methods

5.2 Likelihood, Priors, and Posterior Distributions

5.3 The Metropolis-within-Gibbs Algorithm

5.4 Sequencing and Interval Estimation via MCMC

5.5 Other Inferences

5.6 Example: Sequencing using the MCMC Approach with Simulated Data

6 Some Applications

6.1 Sequencing Drafts

6.1.1 Netflix Paper Drafts

6.1.2 “One Art” Poem Drafts

6.1.3 Versions of “The Song of Solomon”

6.2 Sequencing DEEDS Documents

6.2.1 Example: Comparing SSE and ADP Criterion Functions for Sequencing Grant Documents

6.2.2 Example: Sequencing Grant Documents with Training and Test Sets Chosen Systematically
6.2.3 Example: Sequencing and MCMC Inference for Grant Documents using a Training and a Test Set .................................................. 130

6.2.4 Example: Comparing the Closest Peers, NNLS, and MCMC Methods for Sequencing DEEDS Grants ............................................. 136

6.3 A Method for Document Classification ........................................ 138

6.3.1 Classifying DEEDS Documents .............................................. 139

6.3.2 Example: Classifying DEEDS Documents using Tableaux Distances ................................................................. 140

6.3.3 Example: Classifying DEEDS Documents using LSA-based Distances ................................................................. 144

6.4 Summary of Results for Real Data Sets ........................................ 148

7 Complements and Concluding Remarks ................................. 150

7.1 Summary of Methods and Possible Improvements ..................... 150

7.2 Open Problems and Future Work ............................................ 152

7.3 Potential Applications ........................................................... 153

Appendices ................................................................. 156

A An Example Using the Tableaux Method .................................. 156

B DEEDS Charter Format and Computer Programs ..................... 159

Bibliography ............................................................... 162
Chapter 1

Introduction

Text Mining and Document Retrieval techniques are widely used to gain information from collections of documents. In particular, such methods have been popular for documents on the World Wide Web and are used for a variety of different problems such as: classification (assigning a class/category to each document based on a specified attribute such as the author, subject, document type, or date of publication); query matching (finding the most relevant documents that match a user’s query and ranking those documents by relevance as an internet search engine would do); indexing (describing a document by a set of index terms or symbols); and document summarization (automatic summarization of the key points in a document).

This thesis concerns a related although different problem in which one is given a corpus of \( n \) text documents \( D = \{ D_1, D_2, \ldots, D_n \} \) more or less all on the same topic but with unknown dates. We are interested in estimating automatically the temporal order in which these documents were written using only the words appearing in the corpus. We propose various distance-based approaches for sequencing documents, including some original work that has not been suggested in the literature. This work was originally motivated by the Documents of Early England Data Set (DEEDS), a collection of undated medieval property exchange charters written in Latin and housed at the University of Toronto. However, the methods described in this thesis may be useful for other
applications, such as ordering drafts or versions of works written by an author, ordering different documents written on the same topic and/or written by the same author, etc. Such applications will be examined with specific data sets in Chapter 6. Furthermore, temporal sequencing methods may potentially be useful to sequence events on the internet, determine which blog initiated a thread, follow news stories, and have many other potential applications. Such applications can be found in many fields such as Linguistics, Genetics, and Communication Theory.

1.1 Text Processing Preliminaries

In this section, we explain some basic text processing procedures and define some preliminaries and common terms that are relevant to text mining.

Suppose that we have a corpus of \( n \) documents \( D = \{D_1, D_2, \ldots, D_n\} \) with \( m \) unique words/terms \( W = \{w_1, w_2, \ldots, w_m\} \) that appear in the corpus. Before the corpus can be analyzed, it undergoes a text processing stage in which the documents are cleaned and preprocessed. Each document is parsed (punctuation and capitalization is removed) to reduce each document to a sequence of words/terms. Sometimes stemming is also done to remove the endings of words so that for example, ‘running’, ‘ran’, ‘runners’ all become the root/stem word ‘run’. Stop words/non-contextual words are words that are not important to the subject matter such as prepositions or articles like ‘and’, ‘if’, ‘but’, etc. Such words should likely appear with approximately equal frequency over time and are not considered to be important factors in estimating the order in which the documents were written or for any other documental retrieval purposes. Commonly, stop words are removed from the documents in the text processing stage. However, if one wishes to examine substrings of words in the documents, rather than just individual word occurrences, stop words will sometimes not be removed, as described below.

Suppose a document \( D_i \) consists of words in the order \( (w_1, w_2, \ldots, w_{n_i}) \), where the words may not necessarily be distinct. A shingle is a substring (i.e. set of consecutive words) that appears in a document, and the shingle length/shingle order is defined to be the number of words that
appear in that particular shingle (Broder, 1998). An \( \ell \)-shingle is a shingle of length \( \ell \) (i.e. a set of \( \ell \) consecutive words in a document). For example, a 1-shingle is a single word, a 2-shingle is a pair of words that appear together, etc.

In most document retrieval methods, individual words, rather than shingles, are considered and the documents are represented by vectors. Documents can be denoted by vectors using various methods. The simplest way is to represent document \( i \) by a vector of counts \( x_i = (x_{i1}, x_{i2}, \ldots, x_{im}) \); where each entry \( x_{ij} \) is the number of times the word \( w_j \) appears in document \( i \). Here, \( j = 1, 2, \ldots, m \) ranges over the words in the entire corpus and \( i = 1, 2, \ldots, n \) ranges over the documents in the corpus. Thus, each document in the corpus is represented by an \( m \)-vector where \( m \) is the number of unique words in the corpus as described above. Alternatively, documents could be represented by ‘feature vectors’ or by other vectors using more elaborate techniques (see Section 2.3.3 for details).

Once the documents have been represented by vectors, comparisons between the documents can be done via comparison of their corresponding vectors. This is convenient for calculating distances or similarities between documents, which will be discussed in detail in Chapter 2.

In the next section, we give a brief overview of orderings since we are interested in ultimately estimating the temporal ordering (of the documents) and comparing it to the true ordering when known.

1.2 On Orderings

Once an ordering of a set of elements has been estimated by some sequencing method (for example, by one of the methods to be discussed in following chapters), it is then of interest to measure the accuracy of this estimated ordering. First, we introduce some required notation for orderings.

Suppose we have a set of \( n \) elements to be ordered. Without loss of generality, we will label each element by its position in the true ordering so that the true order of the elements is represented by \( \ell = 1, 2, \ldots, n - 1, n \). Then, any ordering will simply be a permutation of \( \ell \). Let \( \tau(\ell) \) represent such
an ordering. \(\tau(\ell)\) can be represented as \(b_1, b_2, \ldots, b_n\), where \(b_i\) is the true position of the element that was ranked \(i\)th by the permutation \(\tau\). For example, suppose we have 5 elements so that \(\ell = 1, 2, \ldots, 5\) is the true order. Let \(\tau_1\) and \(\tau_2\) represent two different permutations so that \(\tau_1(\ell)\) and \(\tau_2(\ell)\) are the respective orderings induced by these permutations. The table below displays the values for \(b_i\) for two particular cases:

<table>
<thead>
<tr>
<th>(\ell)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>(b_1)</td>
<td>(b_2)</td>
<td>(b_3)</td>
<td>(b_4)</td>
<td>(b_5)</td>
<td></td>
</tr>
<tr>
<td>(\tau_1(\ell))</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>(\tau_2(\ell))</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1.1: An example of orderings

In the case of ordering \(\tau_1\), we have \(b_1 = 1\), which means that \(\tau_1\) has ranked the first element in the correct position, \(b_2 = 3\) indicates that \(\tau_1\) has ranked the 3rd element to be in the 2nd position, \(b_5 = 2\) means that \(\tau_1\) has ranked the 2nd element to be in the 5th position, and so on. On the other hand, for \(\tau_2\), none of the \(b_i = i\) meaning that \(\tau_2\) has not ranked any of the elements in their correct position. However, the ordering of elements 2 to 5 is correct and 1 is the only element that is misplaced in \(\tau_2\).

Below we summarize some commonly used measures between orderings:

1. **Euclidean Distance (ED):**

   We can consider the true order \(\ell = (1, 2, \ldots, n)\) and a given ordering \(\tau = (b_1, b_2, \ldots, b_n)\) as points in an \(n\)-dimensional Euclidean space. Then, the Euclidean distance between the two points is defined as usual:

   \[
   ED(\tau, \ell) = \sqrt{\sum_{i=1}^{n}(\tau_i - \ell_i)^2} = \sqrt{\sum_{i=1}^{n}(b_i - i)^2}.
   \]

   For convenience, the squared Euclidean distance is usually used.
2. **Manhattan Distance** (MD):

The Manhattan distance is defined as the sum of the absolute values of the differences between $\ell_i$ and $\tau_i$:

$$\text{MD}(\tau, \ell) = \sum_{i=1}^{n} |\tau_i - \ell_i| = \sum_{i=1}^{n} |b_i - i|.$$ 

3. **Spearman Rank-Order Correlation** ($\rho$): The Spearman rank-order correlation measures how well the relationship between two variables ($X$ and $Y$) can be described by means of a monotonic function. It is the Pearson correlation coefficient between the ranked variables. Each value is first converted to its rank (denoted by $x_i$ and $y_i$), where the highest value is ranked 1 and the lowest value ranked $n$. Then the correlation is computed from these ranked values:

$$\rho(\tau, \ell) = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2(y_i - \bar{y})^2}}.$$ 

In the case of orderings, a perfect correlation of +1 (or −1) means that the two orderings are the same (or that one is the reversal of the other). A negative value indicates that one ordering is in reverse chronological order to the other whereas, a positive value indicates that the two orderings follow the same direction.

4. **Kendall's Tau** (I):

Kendall’s Tau measures the number of pairwise disagreements between two rankings. Any pair of elements $(i, j)$ is said to be concordant or in agreement if $i$ is ranked higher than $j$ or if $i$ is ranked lower than $j$ in both rankings (i.e. $\tau^{-1}(i) > \tau^{-1}(j)$ and $\ell^{-1}(i) > \ell^{-1}(j)$ or $\tau^{-1}(i) < \tau^{-1}(j)$ and $\ell^{-1}(i) < \ell^{-1}(j)$). The pair $(i, j)$ is said to be disconcordant or disagree if $\tau^{-1}(i) > \tau^{-1}(j)$ and $\ell^{-1}(i) < \ell^{-1}(j)$ or $\tau^{-1}(i) < \tau^{-1}(j)$ and $\ell^{-1}(i) > \ell^{-1}(j)$. Then, Kendall’s correlation coefficient is defined as:

$$I(\tau, \ell) = \frac{(\text{number of concordant pairs}) - (\text{number of disconcordant pairs})}{\frac{1}{2}n(n - 1)}.$$ 

The denominator counts the number of distinct (unordered) pairs. A perfect agreement between the rankings yields a Kendall’s Tau value of 1, whereas a perfect disagreement (one
is the reverse of the other) results in a value of -1. If the two rankings are independent, we would expect a correlation near 0.

If order reversal is considered to be indistinguishable (as is the case in our work), the measures in 1 and 2 should be computed as the minimum of the distance between \( \tau \) and \( \ell \) and the distance between \( \tau \) and the reverse of \( \ell \). It is not necessary to do so for the correlation measures as the sign of those measures will indicate whether the two orderings are in reverse order of one another or not. Below are sample calculations comparing the various distance measures between the two different orderings from the example of Table [1.1] above and the true ordering \( \ell = (1, 2, 3, 4, 5) \):

<table>
<thead>
<tr>
<th>Ordering</th>
<th>Squared Euclidean</th>
<th>Manhattan</th>
<th>Spearman Correlation</th>
<th>Kendall’s Tau</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tau_1 = (1, 3, 5, 4, 2) )</td>
<td>( ED(\tau_1, \ell) = 14 )</td>
<td>( MD(\tau_1, \ell) = 6 )</td>
<td>( \rho(\tau_1, \ell) = 0.3 )</td>
<td>( I(\tau_1, \ell) = 0.2 )</td>
</tr>
<tr>
<td>( \tau_2 = (2, 3, 4, 5, 1) )</td>
<td>( ED(\tau_2, \ell) = 20 )</td>
<td>( MD(\tau_2, \ell) = 8 )</td>
<td>( \rho(\tau_2, \ell) = 0 )</td>
<td>( I(\tau_2, \ell) = 0.2 )</td>
</tr>
</tbody>
</table>

Table 1.2: An example comparing the accuracy between two orderings

We observe that \( \tau_1 \) has lower Euclidean and Manhattan distances from \( \ell \) than does \( \tau_2 \) from \( \ell \) and also that the Spearman and Kendall’s Tau correlation values of \( \tau_1 \) with \( \ell \) are higher than those of \( \tau_2 \) and \( \ell \). This would lead one to believe that \( \tau_1 \) is a better estimate of the ordering than is \( \tau_2 \) (i.e. that \( \tau_1 \) is closer to the true ordering than is \( \tau_2 \)). But in some sense, one could argue that \( \tau_2 \) is in fact better as it has only misplaced one element (the first element) and the order of the other elements is correct. However, since these measures are rank measures for orderings they do not take this aspect into consideration. It is in fact very difficult to define an absolute measure to compare orderings. In this thesis, we will usually use the squared Euclidean distance and the Spearman Rank correlation measures as they are the most commonly used and have been known to perform better than others.

Refer to Huang and Ling (2005) for further details on rank measures for orderings and their comparisons, and Diaconis (1988) for comparisons on permutations.

We now briefly discuss some of the work most related to sequencing documents that has been done
in the past.

1.3 Past Research

There has been a good deal of past research on other text mining problems such as sorting documents by language or topic, query matching, and indexing documents. To date, however, very little work has been done on the specific problem of sequencing documents. The existent few papers on this subject use methods such as time series, which requires specific distributional assumptions and lengthy time series to obtain accurate results (Dalli and Wilks, 2006) or focus on ranking models which are not quite the same as ordering (Cohen, Shapire, and Singer, 1999). Some related work has also been done on calendaring or dating documents (Feuerverger et al, 2008 and Tilahun, 2011). However, our primary interest here is in sequencing not dating and we will therefore need to use entirely different statistical methods. Below is a brief summary of the most relevant papers involving the topic of ordering documents:

Cohen, Shapire, and Singer (1999) examine the problem of ranking elements based on a set of preference judgements (i.e. statements that one element should be ranked ahead of another). Here, the orderings are actually rankings because a particular element is considered to be “better” or preferred in some sense to another element if that particular element is ordered ahead of another. Examples of such types of problems include ranking webpages according to relevance for a particular query, prioritizing unread emails via a certain filter, or even collaborative filtering applications such as the Netflix problem, etc. For this context, Cohen et al propose a two-stage approach to ordering the elements. First, a preference function (i.e. a function indicating how certain it is that one element is preferred or ranked ahead of another) is learned by using an online weight algorithm that is based on the “Hedge” algorithm (Freund and Schapire, 1997) and then the preference of each pair of items is evaluated. Then the learned preference function is used to order the elements in such a way that the estimated ordering agrees as much as possible with the pairwise preferences. The authors show that the problem of finding a total order that agrees with the preference function is NP-complete. However, there are greedy algorithms that can be used to obtain an approximation
to the best ordering. The details are described below:

Let $D$ be the set of $n$ documents/items \{D_1, D_2, \ldots, D_n\} containing $m$ unique words/terms \{w_1, w_2, \ldots, w_m\}. A preference function $PREF(u, v)$ is a binary function $D \times D : [0, 1]$ that returns a numerical measure of how certain it is that item $u$ should be ranked before item $v$. If $PREF(u, v)$ is close to 1 (respectively 0) then there is a strong recommendation to rank $u$ ahead of (respectively after) $v$. A value close to $\frac{1}{2}$ is interpreted as not making any recommendation. A preference function is usually thought of as a linear or boolean combination of primitive preference functions $R_i$. The authors introduce a special type of preference function, $R_{f_i}$, which is called a rank ordering and is defined by an ordering function, $f_i$, as follows:

$$R_{f_i}(u, v) = \begin{cases} 
1, & f_i(u) > f_i(v) \\
0, & f_i(u) < f_i(v) \\
\frac{1}{2}, & \text{otherwise}
\end{cases}$$

- $R_{f_i}(u, v) = 1$ is a strong recommendation that $u$ should be ranked higher than $v$
- $R_{f_i}(u, v) = 0$ is a strong recommendation that $u$ is ranked below $v$
- $R_{f_i}(u, v) = \frac{1}{2}$ is interpreted as an abstention from making a recommendation ($u$ or $v$ or both are unranked).

For example, if we wanted to order documents based on the words they contain, we could consider $f_i(u)$ to be the number of occurrences of the word, $w_i$, in document $u$ for $i = 1, 2, \ldots, m$. Then, $R_{f_i}$ will prefer $u$ to $v$ whenever $w_i$ occurs more often in $u$ than $v$. Another example is one in which we wish to combine the rankings of several web search engines on some particular query. Suppose there are $N$ search engines. Then one may define an ordering function, $f_i$ so that $R_{f_i}$ will prefer webpage $u$ to $v$ whenever $u$ is ranked ahead of $v$ in the list, $L_i$, resulting from search engine $i$, for $i = 1, 2, \ldots, N$. In this case, one could define $f_i(u) = -k$ where $k$ is the position of webpage $u$ in the ranking given by search engine $i$, and let $f_i(u) = -M$ where $M > |L_i|$ for any webpage $u$ not appearing in the list $L_i$. 

Now, in order to obtain a good linear combination of these ordering functions, it is assumed that a set of ranking experts, each of which generates an ordering function, is available. In the case of the second example given above, each ranking expert is a function that submits the user’s query to a different search engine, the elements to be ordered are the set of all webpages returned by the search engines, and the ordering function corresponding to each ranking expert \( i \) is same as given above (i.e. \( f_i(u) = -k \) for the \( k \)th webpage \( u \) returned by search engine \( i \) and \( f_i(u) = -M \) where \( M > |L_i| \) when search engine \( i \) does not return webpage \( u \)). For convenience, we will abbreviate \( R_{f_i} \) as \( R_i \) (i.e. \( R_i \equiv R_{f_i} \)). A weight allocation algorithm that uses the primitive preference functions \( R_i \) is used to “learn” a preference function of the form: \( \text{PREF}(u, v) = \sum_{i=1}^{N} \omega_i R_i(u, v) \). Cohen et al have adopted the online learning structure by Littlestone (1988) in which learning occurs in sequences of rounds and the weights are incrementally updated. On each round \( t \), \( D^t \) is the set of elements to be ranked for which each ranking expert \( i \), for \( i = 1, 2, \ldots, N \) provides an ordering function \( f^t_i \) and each ordering function induces a preference function, \( R^t_i \). For instance, in the query search example from above, \( f^t_i \) is the ordering function corresponding to the list \( L^t_i \) of webpages returned by the \( i \)th ranking expert for the \( t \)th query, and \( D^t \) is the set of all webpages appearing in any of the lists \( L^t_1, L^t_2, \ldots, L^t_N \) returned by the search engines for query \( t \). In addition, on each round, feedback, \( F^t \) is available from the user. Feedback is assumed to be a set of pairs \((u, v)\), each corresponding to an assertion “\( u \) should be preferred to \( v \)”. For example, \( F^t \) could be the user’s set of preferences between the webpages returned for the \( t \)th query or could be measured indirectly by the amount of time spent on each webpage returned for query \( t \), etc. Weights are updated based on the “Hedge” algorithm (Freund and Shapire, 1997) as follows:

First, a loss of a preference function with respect to a user’s feedback is defined as:

\[
\text{Loss}(R, F) = \frac{\sum_{(u, v) \in F} (1 - R(u, v))}{|F|} = 1 - \frac{1}{|F|} \sum_{(u, v) \in F} (1 - R(u, v)).
\]

This loss function has a probabilistic interpretation. If \( R \) is viewed as a randomized prediction which predicts that \( u \) will be ranked higher than \( v \) with probability \( R(u, v) \), then \( \text{Loss}(R, F) \) is interpreted as the probability of \( R \) disagreeing with the feedback on a pair \((u, v)\) chosen uniformly
at random from $F$.

Let $\omega^t = (\omega_1^t, \ldots, \omega_N^t)$ denote the values of the positive weight vector at round $t$. Initial values of $\omega_i^1 = 1/N$ for $i = 1, 2, \ldots, N$ are chosen if there is no prior information about the ranking experts. On each round $t$, the weight vector is used to combine the preference functions from the different ranking experts into a preference function: $\text{PREF}^t(u, v) = \sum_{i=1}^{N} \omega_i^t R_i^t(u, v)$. The weight vector is updated on each round using the following multiplicative rule:

$$
\omega_i^{t+1} = \frac{\omega_i^t \beta^{\text{Loss}(R_i^t, F_i^t)}}{Z^t};
$$

where $\beta \in [0, 1]$ is a parameter and $Z^t$ is a normalization constant chosen so that the weights sum to 1 after the update (i.e. $\sum_{i=1}^{N} \omega_i^{t+1} = 1$). Therefore, on each round the weights of the ranking experts are adjusted so that ranking experts producing preference functions that mostly agree with the feedback are increased.

Now that the preference function has been learned, the second stage of the process is to estimate a total order, $\rho_{\text{opt}}$, that best agrees with the learned preference function. This is done by finding the ordering $\rho$ that maximizes the agreement between the learned preference function and any ordering $\rho$:

$$
\text{AGREE}(\rho, \text{PREF}) = \sum_{u, v : \rho(u) > \rho(v)} \text{PREF}(u, v).
$$

So, $\rho_{\text{opt}}$ will be the total ordering that maximizes the above function $\text{AGREE}(\rho, \text{PREF})$. This optimization problem is NP-complete even if the preference function is a linear combination of simple well-behaved functions. Thus, Cohen et al propose a greedy algorithm for estimating the total order. The operation of this algorithm is easily demonstrated by means of a directed weighted graph, where the set of vertices are the elements to be ordered, with edges $u \rightarrow v$ that have weights $\text{PREF}(u, v)$. For each element $v \in D$ first $\pi(v) = \sum_{u \in D} \text{PREF}(v, u) - \sum_{u \in D} \text{PREF}(u, v)$ is calculated. Let $t = \arg \max_{u \in D} \pi(u)$ (the element with the most potential to be ranked the highest), and assign $t$ the maximal rank. Then, $t$ and all its incident edges are removed from the graph and the values of $\pi(v)$ for the remaining vertices are updated. The above process is repeated...
until the graph is empty. Deleted elements on each iteration will be assigned a smaller rank than that of the previous iteration. The authors demonstrate that their proposed greedy algorithm finds a good approximation to the optimal total ordering. Note that it is possible in this method to leave some elements unranked as the preference functions allow for it.

**Dalli and Wilks (2006)** use time series and filtering methods to predict the time of creation/writing of a set of documents. The main assumption in their work is that word usages vary over time and follow certain patterns. Some terms are particularly popular for a certain period of time and then fade off (fads, obsolete words), while other words are commonly used throughout time (usually stop words such as ‘if’, ‘and’, ‘but’, etc). Also, current events and issues that occur during the time an author writes the document will affect their word choices. There are certain words that we know are unlikely to have been used before a certain date (for example, the word “blog” used before 1999) and these trends and word usages can give temporal information. The authors provide a method for using periodic temporal information that can be used to create temporal association rules. Ongoing research has created tagging methodologies to classify temporal expressions.

In their work, Dalli and Wilks use a SARIMA (Seasonal Auto-regressive Integrated Moving Average) model for each term in the corpus, based on training documents having known dates. The SARIMA adds seasonal auto-regressive and moving average polynomials that can manage seasonally-varying data in time series. Each series is then decomposed into 2 parts: a periodic component that repeats itself in a predictable manner, and a non-periodic component (which is left over after filtering out the periodic component). The periodic component is utilized to isolate temporal association rules between terms and seasons/time periods. For example, to determine if a term occurs most frequently on a weekly basis, a window size of 7 is applied to the series and then it is verified if the periodic time series always has a peak within this window. Each association rule has the following information: term ID, period type (day, week, month, quarter, year), and period number and score matrix. The period number and score matrix represent a probability distribution for the likelihood of a term appearing on a certain period number. For example, the score matrix for the term “Monday” would have a high score for period 1 when the period type is set to day. These
association rules are then used to estimate the likelihood of terms appearing in the documents for
the evaluation set and the documents are automatically dated as follows: The probability density
function (pdf) for each term is first weighted and then added up. The inverse-document frequency
(idf) is used to weight each pdf so that terms that occur in only a few documents are weighted
higher than those that occur in almost all documents. (See Section 2.3 for further details on idf.)
Periods that obtain the highest score are ranked for each type of period and two guesses per period
type are made for each document (i.e. for each document, 2 guesses are made each for its day of
the week, week number, month number, quarter, and year for a total of 10 guesses).

When this system was run on the LDC GigaWord corpus (LDC, 2003) and evaluated, they found
that the week classifier performed the most poorly. However, the results from all period types
can be used to check the accuracy of the classifiers and correct the estimated date of creation.
For example, if a particular document had the following results: Day = 3 (Wednesday), Week =
52, Month = 1 (January), Quarter = 1, Year = 2006, it is obvious that the week number 52 is
inconsistent with the rest of the period type classifications. The system estimated the document
to be written in the last week of the year rather than at the beginning of the year even though the
month was estimated correctly. This most likely occurred because week 1 comes after week 52 (if
the weeks are viewed as a cycle) and so the system can use this information along with the other
classifiers to properly date the document.

The disadvantage of this technique is that SARIMA models require fairly long series (at least
50 data points) before getting accurate results. Also, the non-periodic component is not used to
extract association rules in this paper. However, the non-periodic data could be used to retrieve
information about terms that occur less frequently in the corpus in order to refine the association
rules. This should be done with caution as the rules derived from the non-periodic data may pertain
to the specific characteristics of the documents used to train the system. Dalli and Wilks indicate
that research on this topic is ongoing.

Feuerverger et al (2008) and Tilahun (2011) studied the problem of calendaring (i.e. dating)
historical documents based on the words they contain. They created two different methods for calendaring documents: distance based methods and the Maximum Prevalence method. A training set, $\mathcal{T}$, of documents which have known dates are used to estimate the unknown dates of the documents in the validation set.

In the first method, resemblance distances (as defined in Section 2.1) based on shingles of different orders are calculated between a document with an unknown date and each document in the training set. The authors introduce different types of ‘correspondence distances’ that result from applying specific forms of functions to the resemblance distances, and the advantages of each type is discussed. The appropriate distance measure for a specified shingle order $k$ that will be used is denoted as $d_k(i,j)$ where document $i$ is the document to be dated and $j \in \mathcal{T}$. Then, a kernel weight function is defined with different bandwidths for each shingle order on the dates for the documents in the training set based on their distances to the document that is to be dated, as follows:

$$a(i, j) = a(i, j|h_1, \ldots, h_r) = \prod_{k=1}^{r} K_{h_k}(d_k(i,j)).$$

In the above, $K$ is a non-negative, non-increasing kernel function, and $h_1, \ldots, h_r$ are bandwidths, and shingle orders from 1 to $r$ (for a specified $r$) are all used. Cross-validation is used for bandwidth selection. Each of these weights $a(i, j)$ measure the closeness of the document to be dated with each document in the training set. The date of the document is then estimated using a locally constant kernel regression estimator on the weights above as follows:

Suppose the training set has $n$ documents. The response is considered as $Y_j = t_j$ for $j \in \mathcal{T}$ and the model is $(t_1, t_2, \ldots, t_n)' = (1, 1, \ldots, 1)' + (\epsilon_1, \epsilon_2, \ldots, \epsilon_n)$ with covariance matrix $W = \text{diag} \left[ \frac{1}{a(i,1)}, \frac{1}{a(i,2)}, \ldots, \frac{1}{a(i,j)}, \ldots, \frac{1}{a(i,n)} \right]$. The weighted least squares estimate of $t_i$, the date for document $i$ (where $i \notin \mathcal{T}$), is given by:

$$\hat{t}_i = \arg \min_t \sum_{j \in \mathcal{T}} (t_j - t)^2 a(i, j) = \frac{\sum_{j \in \mathcal{T}} t_j a(i, j)}{\sum_{j \in \mathcal{T}} a(i, j)}.$$
The second method for dating documents is called the Maximum Prevalence method. Let \( s_k(D) \) denote the set of shingles of order \( k \) that appear in document \( D \), \( n_s(D) \) represent the number of times shingle \( s \) occurs in \( s_k(D) \), and \( N(D) \) be the total number of elements in \( s_k(D) \). In this method, the probability of the occurrence of each shingle over time is estimated using a local constant polynomial logistic regression model. (The occurrence of one shingle in a specific document is considered to be independent from any other shingle in that document.) For each shingle, \( s \), an estimate of its probability of occurrence as a function of time, denoted \( \pi_s(t) \), is required. In order to do this, a logistic model is used. It is assumed that the conditional distribution of \( n_s(D) \) given \( N(D) = m \) and \( t_D = t \) has a binomial distribution:

\[
\mathcal{L}(n_s(D)|N(D) = m, t_D = t) \sim \text{Bin}(m, \pi_s(t)).
\]

Then,

\[
\hat{\pi}_s(t) = \frac{\sum_{i \in T} n_s(D_i)K_h(t_{D_i} - t)}{\sum_{i \in T} N(D_i)K_h(t_{D_i} - t)}.
\]

Next, the probability of occurrence of a document \( D \) as a function of time is defined as follows:

\[
\pi_D(t) = \prod_{s \in s_k(D)} \pi_s(t) \prod_{s \notin s_k(D)} (1 - \pi_s(t)).
\]

Note that this probability is based on both shingles that occurred in \( D \) and those that did not appear in \( D \) but did appear in the training set. Then \( \pi_D(t) \) is estimated by \( \hat{\pi}_D(t) = \prod_{s \in s_k(D)} \hat{\pi}_s(t) \prod_{s \notin s_k(D)} (1 - \hat{\pi}_s(t)) \) and finally the date of document \( D \) is estimated by:

\[
\hat{t}_D = \arg \max_R \hat{\pi}_D(t);
\]

where \( R \) is the time range over which the documents were written. This was a known time range in the case of the DEEDS data set that the authors worked with. The authors show that the second factor in \( \pi_D(t) \) is immaterial using an approximation. Therefore, the most likely date in which a document was written is estimated by maximizing the “prevalence” of all the shingles that
make up that particular document. Here, the prevalence is a likelihood function, which by the independence assumption is just the product of the individual probabilities of occurrence for each shingle. Optimal bandwidths can be selected for each shingle using leave-one-out cross-validation procedures or by using the validation set. Tilahun et al found that shingles of order 2 seemed to be the most useful in predicting dates.

Other references that deal with temporal ordering issues include Blei and Lafferty (2006), and de Fong, Rode, and Hiemstra (2005). In the next section, we outline our new approach to sequencing documents.

1.4 New Approach

As already mentioned, much research has been done in the field of document retrieval, but not much work has focused on the aspect of temporal ordering of documents. However, methods for comparing documents similar to those used in document retrieval can still be considered for such sequencing purposes. This is because one would expect time separation between documents to have some degree of correlation with aspects of the similarity between those documents, so that we can try to use a distance-based approach to sequencing documents. Our approach relies on the assumption that documents written closer together in time will be more similar in content. We propose, in the first instance, the following two-step automated procedure:

1. Obtain distance measures between pairs of documents (using only their word characteristics)

2. Estimate an optimal ordering of the documents based on these pairwise distance measures

Various ways of obtaining distance measures between documents are discussed in Chapter 2 and techniques for sequencing elements based on such distances are described in Chapters 3 and 5.

As mentioned before, our work is different from that of Cohen et al as they were interested in ranking documents, which provides a sense of one document being better or more preferred to
another. In our case, we are interested in ordering the documents in a time sequence, not rank ordering. However, both methods are similar to the extent that both use pairwise measures between documents to find an overall ordering that best agrees with the pairwise measures. Cohen et al use preferences between documents, whereas we will use distance measures between document pairs. Also, note that preference functions are not symmetric, but distance measures are. Another similarity is that we will also use a greedy algorithm (to be described in Section 4.1) to estimate an optimal ordering although it is based on an entirely different approach and does not use graphs.

The work of Dalli and Wilks, Feuerverger et al, and Tilahun are related to our work as their problem deals with estimating dates using word patterns in the documents. However, we will not use time series or any filtering methods here as did Dalli and Wilks. Also, we will use entirely different statistical tools and methodologies from the work of Tilahun. However, both Tilahun’s and our work was motivated primarily by the DEEDS data set and we note that there are other similarities such as the use of distance measures between documents, even though we are using different types of measures.

1.5 Outline Of Thesis

The remainder of this thesis is described as follows: Chapter 2 outlines various methods for obtaining distance and similarity measures between pairs of documents. Some of the distance measures introduced are ones that are commonly used in document retrieval, while some of the methods proposed are original. We also suggest some techniques to attempt to find distance measures that will have a reasonably high correlation with time separation and propose some new ideas that are extensions of previous methods. Chapter 3 describes our method of sequencing elements based on using an appropriate regression model on the pairwise distances between elements. Matrices and properties of the estimators resulting from this model are detailed and then we describe some more complex variations of the model that could be applied to sequencing problems. We also describe a “Closest Peers” approach to sequencing which does not impose any model assumptions on the pairwise distances between elements. Chapter 4 focusses on optimization algorithms for sequencing
problems. We propose a specific greedy algorithm for estimating the ordering that will optimize a
given criterion function along with some variations to the algorithm. In Chapter 5 we adopt an
MCMC approach to estimating the ordering of a set of elements based on pairwise distances (an
alternate to the sequencing methods from Chapter 3) and to making specific inferences about the
true ordering. An important advantage of this method is that it allows for a natural path to inference on permutations. Chapter 6 applies the methods described in this thesis to real collections of
documents along with the interpretation and discussion of the results and methods used. Finally,
in Chapter 7 we discuss the scope of this problem and suggest some different (non-distance based)
approaches to sequencing documents. We also point out some of the remaining open problems and
possibilities for future work.
Chapter 2

Distance Measures

Developing appropriate distance measures between pairs of documents is essential for us since our main approach to sequencing is based on distances. Many types of distance measures between documents are available, however only some of them will be useful for the problem of sequencing. This is because only certain types of distances have some degree of correlation with the time separation among documents, leading to adequate sequencing results. Other distances may not be useful for explaining time order, but may still allow us to explain other factors relevant to the corpus, such as document subject, style of writing, etc.

The first section of this chapter summarizes some of the common distance and similarity measures that can be computed between pairs of documents based only on word frequencies. (Distance and similarity are reciprocal concepts.) We then discuss some of the reasons why it is necessary to apply more sophisticated methods that take into account the entire corpus (rather than only the word counts and/or other measures involving only pairs of documents). Latent Semantic Analysis (LSA) is described in Section 2.3 including the basic technique, the Singular Value Decomposition (SVD) method, how to obtain distances, how LSA is used for information retrieval purposes, as well as its advantages and limitations. We also explore some new ideas relating to SVD in Section 2.4 such as possible methods to find distances that align with time (2.4.1), and Logistic SVD models (2.4.2). We then propose some new measures based on a certain “Tableaux Method” that considers
important features of the corpus as a whole in order to try to find effective distances in Section 2.5. Finally, since some of the methods we discuss result in obtaining similarities rather than distances, we discuss the topic of converting similarities to distances in Section 2.6.

2.1 Types of Distance/Similarity Measures

Some approaches result more naturally in measures of similarity between documents whereas others result in measures of distance. The first step to our sequencing approach involves finding distance measures between the documents and so it is of interest to compute distances directly or to obtain similarities and then change them into distances. In general, it is difficult to convert from similarity to distance measures. This will be further discussed in Section 2.6.

First, we summarize some common distance and similarity measures between vectors. Recall that any document $i$ can be represented as a (column) vector $x_i = (x_{i1}, x_{i2}, \ldots, x_{iK})'$, for some value $K$, as described in Section 1.1. For example, the dimension of the vector, $K$, can be the number of unique words if using direct frequencies as in Section 1.1. Alternatively, it can be the number of features of the corpus taken from a Singular Value Decomposition as described in Section 2.3.3. Below are some measures between documents $a$ and $b$:

1. **Metric Distances**

   The Minkowski Distances $L_p$ are metrics.

   $$L_p(x_a, x_b) = \left( \sum_{j=1}^{K} |x_{aj} - x_{bj}|^p \right)^{1/p}.$$

   For $p = 1$, this is sometimes called the *Manhattan distance*, and for $p = 2$, it is the usual *Euclidean distance*. 
2. **Cosine Similarity**:

*Cosine Similarity* is defined by the cosine of the angle between vectors:

\[
s^{(C)}(x_a, x_b) = \frac{x_a' x_b}{\|x_a\| \|x_b\|};
\]

where primes denotes transpose and \(\|\cdot\|\) denotes norm.

This measure does not depend on scaling, i.e., \(s^{(C)}(\alpha x_a, \beta x_b) = s^{(C)}(x_a, x_b)\) for \(\alpha, \beta > 0\).

3. **A Correlation Measure**:

The *Normalized Pearson Correlation Similarity* is defined as:

\[
s^{(P)}(x_a, x_b) = \frac{1}{2} \left( \frac{(x_a - \bar{x}_a)'(x_b - \bar{x}_b)}{\|x_a - \bar{x}_a\| \|x_b - \bar{x}_b\|} + 1 \right);
\]

where \(\bar{x}\) is the average feature value of \(x\) over its dimensions.

This measure takes values in \([0, 1]\).

4. **Jaccard Similarity Measures**:

The *Binary Jaccard Coefficient* measures the ratio of the number of shared words (or terms) of documents \(a\) and \(b\) to the number of unique terms appearing in documents \(a\) or \(b\). Let \(K\) be the number of unique words that appear in the corpus. Each document is represented by a binary \(K\)-dimensional vector indexing whether or not the term corresponding to the entry of the vector appears in that particular document. The sets \(A\) and \(B\) correspond to the words in documents \(a\) and \(b\) respectively. The Binary Jaccard Coefficient is defined as:

\[
s^{(B)}(A, B) = \frac{|A \cap B|}{|A \cup B|}.
\]

Or equivalently, it can be defined in terms of the vectors as:
This measure can be extended for use on general vectors \( x_a \) and \( x_b \) that have continuous or non-negative (non-binary) features by using the "Extended Jaccard Similarity":

\[
s^{(J)}(x_a, x_b) = \frac{x'_a x_b}{\|x_a\| \|x_b\| - x'_a x_b}.
\]

This measure is both translation and scale invariant.

5. **Resemblance Distance:**

This is a set theoretic similarity measure that was originally proposed by Broder (1997) in the context of query searches. Recall that an \( \ell \)-shingle is a sequence of \( \ell \) consecutive words from a document (as defined in Section 1.1). Let \( S_\ell(D_a) \) be the set of distinct \( \ell \)-shingles of document \( D_a \). For a fixed shingle order \( \ell \), the **Resemblance** of documents \( D_a \) and \( D_b \) is defined as:

\[
\text{res}_\ell(D_a, D_b) = \frac{|S_\ell(D_a) \cap S_\ell(D_b)|}{|S_\ell(D_a) \cup S_\ell(D_b)|}.
\]

Note: this is an extension of the Binary Jaccard similarity measure defined above using sets \( A \) and \( B \) as the set of distinct \( \ell \)-shingles in documents \( a \) and \( b \) respectively, rather than sets of single words. The distance between \( D_a \) and \( D_b \) is then defined to be:

\[
dis_\ell(D_a, D_b) = 1 - \text{res}_\ell(D_a, D_b).
\]

This distance measure is a metric (i.e. satisfies the triangle inequality).
6. **Levenshtein Distance**:

The Levenshtein distance, which we discuss next, is commonly used in text processing applications when comparing two strings of characters. When comparing character strings, one can consider three types of operations: deletion, insertion, and substitution of letters. One can always transform one string of characters into another by performing some sequence of these operations. In the worst case, one can delete all of the letters and then insert the correct letters needed. A distance between two strings can be defined as the least number of such changes that are required in order to convert one string into the other.

Let \( a = (a_1, a_2, \ldots, a_{K_1}) \) and \( b = (b_1, b_2, \ldots, b_{K_2}) \) be two character strings. The Levenshtein/\textit{Edit Distance} between two strings \( a \) and \( b \) is the minimum number of single-character edits required to change one string into the other, where allowable operations for edits are deletions, insertions, and substitutions of single characters. Recall that each string \( a \) can be represented as a vector \( a \) of the characters it contains. The formula for the Levenshtein distance between \( a \) and \( b \) is given by \( \text{lev}_{a,b}(|a|, |b|) \), where:

\[
\text{lev}_{a,b}(i, j) = \begin{cases} 
\max(i, j) & \text{if } \min(i, j) = 0 \\
\min \begin{cases} 
\text{lev}_{a,b}(i - 1, j) + 1 \\
\text{lev}_{a,b}(i, j - 1) + 1 \\
\text{lev}_{a,b}(i - 1, j - 1) + I(a_i \neq b_j)
\end{cases} & \text{otherwise}
\end{cases}
\]

Here, \( i = 0, 1, 2, \ldots, |a| - 1, |a| \) and \( j = 0, 1, 2, \ldots, |b| - 1, |b| \). Basically, the Levenshtein distance between \( a \) and \( b \) is calculated by filling in a matrix, \( \text{lev}_{a,b} \), where \( i \) and \( j \) in the above formula denote the row and column indices respectively. The formula defines the entries of this matrix as follows: If either \( i \) or \( j \) is 0, the entry is equal to the other index (i.e. if either of the string lengths are 0, the distance is equal to the length of the other string). So, the first column of the matrix will have entries \( 0, 1, 2, \ldots, |a| \) and the first row will contain the entries \( 0, 1, 2, \ldots, |b| \). Otherwise, the entry is given by the minimum of the three expressions in the formula and is computed using the three entries that are directly above it, to the left.
of it, and the entry that is located in the row above and column to the left of it. Note that
the first component of the three minima corresponds to deletion, the second to insertion,
and the third to matching/mismatching depending on whether or not the characters are the
same. The formula is defined recursively on each character of the strings. Once the matrix is
completed, the Levenshtein distance is the last (lower right corner) entry, \( \text{lev}_{a,b}(|a|, |b|) \).

Below is an example of the Levenshtein distance computation between two simple strings
\( a = \text{“rose”} \) and \( b = \text{“horse”} \):

\[
\begin{array}{cccccc}
& h & o & r & s & e \\
0 & & 1 & 2 & 3 & 4 & 5 \\
r & 1 & 2 & 3 & 2 & 3 & 4 \\
o & 2 & 3 & 2 & 2 & 3 & 4 \\
s & 3 & 4 & 3 & 3 & 2 & 3 \\
e & 4 & 5 & 4 & 4 & 3 & 2 \\
\end{array}
\]

In this case, the string lengths are \(|a| = 4\) and \(|b| = 5\). Intuitively, to convert “rose” into
“horse” we need a minimum of two operations: for example, change the ‘r’ to ‘h’ and insert
an ‘r’ between the ‘o’ and ‘s’. From the matrix, the Levenshtein distance is computed as the
lower right corner entry, \( \text{lev}_{a,b}(4, 5) = 2 \).

The Levenshtein distance is actually a metric. It satisfies non-negativity, symmetry, identity
of indiscernibles, as well as the triangle inequality. Furthermore, the following bounds hold:

- \( \text{lev}_{a,b}(|a|, |b|) \geq | |a| - |b| | : \) it is at least the difference of the length of the strings
- \( \text{lev}_{a,b}(|a|, |b|) \leq \max(|a|, |b|) : \) it is at most the length of the longer string

A normalized version of the Levenshtein distance can be computed as:

\[
d\text{Lev}(a, b) = \frac{\text{lev}_{a,b}(|a|, |b|)}{\max(|a|, |b|)}.
\]
It typically makes more sense to consider the normalized version, as the length of the strings tends to matter. For instance, if one change is required to transform a short string into another short string of equal length, then those strings would be considered to be more different from each other than if only a single change is required to transform a long string into another long string of equal length. The normalized Levenshtein measure accounts for such differences whereas the unnormalized distance would be the same regardless of the string lengths.

If one thinks of a document as a string (containing all the words of the document in the order they appear), the Levenshtein distance can be used to compare them. This type of measure is particularly useful when comparing drafts or different versions of a manuscript. It is a natural distance in that context as we would expect authors to insert, delete, and change words as successive drafts of a document are produced.


All of the above distance measures are based on just matching terms or characters for a pair of documents and do not examine the entire corpus simultaneously. Specifically, the above measures 1. to 5. are obtained by comparing the similarity of two documents using direct word frequencies, while the Levenshtein distance is based on character matchings. Alternatively, one could compare documents via more sophisticated techniques which take the entire corpus into account, such as Latent Semantic Analysis, or by new methods such as the Tableaux method that will be introduced in Sections 2.3 and 2.5 respectively.

2.2 Problems with using only Word Frequencies

Measuring distances between document pairs by comparing only word frequencies is potentially problematic. One issue is that such methods treat terms as being statistically independent from one another. Thus, matching (or not matching) of two terms that frequently occur together will be treated the same as would be two terms generally not found together in the same document.
A further drawback is that word frequencies do not differentiate among the usage of the terms and so may not be a good measure of similarity. Specifically, direct word matching methods do not properly deal with synonymy (words that express the same concept) and polysemy (words with more than one distinct meaning). So synonyms will not be detected as describing the same concept (ex. present vs. gift, smart vs. intelligent) and homonyms/homographs would be incorrectly matched (bark - sound a dog makes vs. bark - skin of a tree). Hence, methods based on word frequencies often require the use of dictionaries or vocabularies involving human intervention, which can be expensive and yet not necessarily be effective. Since authors have different word preferences, we often need techniques that capture the meaning and concepts behind the words, not just the words themselves. In this respect, it is also important to regard each document as a part of an entire collection of documents rather than just examining them individually. Thus, we need methods that go beyond lexical word meanings, and that can detect semantic relationships based on the overall patterns of word usage over the whole corpus.

2.3 Latent Semantic Analysis (LSA)

*Latent Semantic Analysis* (LSA) was introduced by Deerwester et al in 1988 and is a popular technique in natural language processing. It examines relationships between documents and the terms they contain. The technique is similar to several other statistical and mathematical procedures such as eigenvalue decomposition, spectral analysis, factor analysis, and multidimensional scaling. LSA is also referred to as *Latent Semantic Indexing* (LSI) in the context of information retrieval.

2.3.1 The Document-Term Matrix

Suppose that a corpus contains $n$ documents $D = \{D_1, D_2, \ldots, D_n\}$ and that within this corpus there are a total of $m$ unique words/terms $\{w_1, w_2, \ldots, w_m\}$, each of which appear in at least 2 of the documents. (If a term only appears in a single document, it cannot be used for comparing documents.) The LSA method starts with an $n \times m$ Document-Term Matrix, $A$, where each entry
\( A_{ij} \) counts the number of times the term corresponding to the \( j \)th column appears in the document corresponding to the \( i \)th row. An optional but common step before proceeding is to transform each entry \( A_{ij} \) in the matrix to its term frequency-inverse document frequency. “Term Frequency-Inverse Document Frequency” (tf-idf) measures how important a particular term is to a particular document after taking into account all of the documents in the collection. The “term frequency” (tf) is the proportion of terms in the document equal to that term, and the “inverse document frequency” (idf) is a measure of the inverse proportion of documents in the corpus that contain that term. The formulas are as follows:

\[
(tf)_{j,i} = \frac{\eta_{j,i}}{||D_i||} \quad \quad \quad (idf)_j = \log \frac{n}{\sum_{i=1}^{n} I(w_j \in D_i)} ;
\]

where, \( \eta_{j,i} = \) number of times the word \( w_j \) appears in document \( i \) and \( I(\cdot) \) is the indicator function.

Finally,

\[
\text{tf-idf}(w_j, D_i) = (tf)_{j,i} \times (idf)_j .
\]

The principle here is that the more times a term appears in a document the greater its importance to that document, however if the term appears in many other documents then its contribution is less significant to the particular document.

### 2.3.2 Singular Value Decomposition and Rank Reduction

For notational convenience, we continue to call our matrix \( A \) even after the elements may have been weighted, for example by the tf-idf, as indicated above. Next, a Singular Value Decomposition (SVD) is performed on \( A \) and the matrix is written as a product:

\[
A = U \Sigma V^T \quad \text{where}
\]
• \( U_\ell \) is \( n \times \ell \), \( V_\ell \) is \( m \times \ell \), \( \Sigma_\ell \) is \( \ell \times \ell \), where \( \ell = \text{rank}(A) \leq \min(n, m) \), which is typically \( = n \).
• \( U_\ell \) and \( V_\ell \) are orthogonal matrices: \( U_\ell^T U_\ell = V_\ell^T V_\ell = I \).
• The diagonal matrix \( \Sigma_\ell \) contains the positive and non-increasing singular values of \( A \):
The singular values are equal to the positive square roots of the eigenvalues of \( AA^T \) or \( A^T A \).
• The columns of \( U_\ell \) (\( V_\ell \)) are called left (right) singular vectors:
The columns of \( U_\ell \) (\( V_\ell \)) are the eigenvectors of \( AA^T \) (\( A^T A \)).

In LSA, the dimensionality of the term-document matrix is then reduced to some value \( K \) by deleting all but the \( K \) largest singular values and their corresponding rows and columns from \( \Sigma_\ell \), resulting in the \( K \times K \) matrix \( \Sigma_K \). The \( \ell - K \) columns of \( U_\ell \) and \( V_\ell \) corresponding to the smallest singular values are deleted to obtain the \( n \times K \) matrix \( U_K \) and the \( m \times K \) matrix \( V_K \). Here, the chosen dimension \( K \) represents the number of latent variables corresponding to the presumed underlying true factors/topics/features of the corpus. Unlike in Factor Analysis, we do not wish to reduce the dimensionality to a very small number, as we are not interested here in visualizing the space. Rather, the goal is to minimize the degree of distortion in the space and to project the documents and terms into a lower dimensional space spanned by these ‘features’. The choice of \( K \) should therefore be large enough to represent the data, but not so large as to include unimportant sampling error or noise in the fit of the data. The value of \( K \) is usually decided empirically; a value of around 100 often appears to work well for thousands of documents.

We recall that the best rank \( K \) approximation of \( A = U_\ell \Sigma_\ell V_\ell^T \) in a least-squares sense is: \( \hat{A} = U_K \Sigma_K V_K^T \). Specifically, the *Eckhart-Young Theorem* states that \( \hat{A} \) is the matrix of rank \( K \) which minimizes the Frobenius Norm of the difference between \( A \) and any other matrix \( B \). The Frobenius Norm of the difference of \( A \) and \( B \) is defined as: \( \| A - B \|_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} (A_{ij} - B_{ij})^2} \). Hence, \( \hat{A} \) defined in this way is the closest matrix to \( A \) in the least squares sense, and so is considered to be the best rank \( K \) approximation to \( A \).
2.3.3 Obtaining Distances/Similarities using LSA

The approximation $\tilde{A}$ to the document-term matrix $A$ contains updated counts that may be thought of as averages (of a kind) over all documents (as opposed to counts with respect to just a particular document). The row vectors of $\tilde{A}$ represent characteristics of the individual documents, while the columns represent the terms. One possible way to measure similarities between documents (terms) is to use the inner products between the row (column) vectors of $\tilde{A}$. The inner product between any two row vectors of $\tilde{A}$ reflects the extent to which two documents have similar terms, and the square $n \times n$ matrix $\tilde{A}^T \tilde{A} = U_K \Sigma_K^2 U_K^T$ contains the pairwise similarities between the documents. Likewise, the inner product between two column vectors of $\tilde{A}$ measures how similar the two terms are, across all the documents in the corpus, and the square $m \times m$ matrix $\tilde{A}^T \tilde{A} = V_K \Sigma_K^2 V_K^T$ contains the pairwise similarities between terms. The matrix of pairwise similarities can then be converted to a matrix of pairwise distances. Converting similarities to distances will be discussed in Section 2.6.

After completing the SVD, each document and each term is represented as a vector in $K$-dimensional space. Specifically, the SVD tries to quantify the $K$ most important features of the documents and of the terms, with row vectors $u_i = (u_{i1}, u_{i2}, \ldots, u_{iK})$, $i = 1, 2, \ldots, n$ of $U_K \Sigma_K$ representing the documents and column vectors $v_j = (v_{j1}, v_{j2}, \ldots, v_{jK})$, $j = 1, 2, \ldots, m$ of $V_K \Sigma_K$ representing the terms. We call $u_i$ and $v_j$ ‘feature vectors’ of the documents and of the terms respectively. Various ways to obtain similarities or distances between documents or terms can be based on these feature vectors. For example, similarities/distances can be calculated by taking Pearson correlations, Euclidean distances, cosine of the angle between any two vectors, or by any reasonable measures between vectors as outlined in Section 2.1. In LSA work, Euclidean distances and cosine similarities between feature vectors are typically used. After obtaining similarities/distances, various approaches may then be taken for information retrieval purposes, as summarized in the next section, or for sequencing documents which is our main goal and which is discussed in Chapter 3.
2.3.4 Using LSA for Information Retrieval

Information Retrieval is mainly concerned with automatic indexing and retrieval methods such as matching documents to a user’s query. Any two objects (documents/terms) that appeared in the original data can be compared using their distances. (Deerwester et al, 1988 and 1990). For example, one can find the closest documents to a particular document, or which terms are the most similar to a particular term, etc. In LSA work, three basic types of relations are of interest: (1) comparing two documents; (2) comparing two terms; (3) comparing a term and a document. Since each document and each term can be mapped to a point in $\mathbb{R}^K$, it is easy to obtain distances between any two documents or any two terms using their feature vectors to measure their proximity.

However, to see how associated a particular term is to a particular document is a slightly different matter, as we are then interested in a specific entry of $\hat{A}$. If we are interested in the association between term $j$ and document $i$, we must compare the $i$th row of $U_K\Sigma_K^{1/2}$ with the $j$th row of $V_K\Sigma_K^{1/2}$. Note that comparisons between a term and a document uses this configuration of points, whereas the within-comparisons (term to term or document to document comparisons) use the rows of $U_K\Sigma_K$ and $V_K\Sigma_K$ (i.e., feature vectors) as coordinates as outlined in Section 2.3.3. Both configurations are similar and they differ only by stretching or shrinking the axes by a factor of $\Sigma_K^{1/2}$.

It is more complicated to find relationships between an object that appeared in the data corpus and one that did not originally exist there (for example, finding documents that match a query or representing new documents that were not a part of the original analysis). Approaches for these types of problems are described in Deerwester et al (1990) and are essentially as follows: A query/new document can be thought of as a “pseudo-document” which can be represented as a vector of terms as done in LSA or as shown in Section 1.1. Suppose we have a new object $q$ and start with its term vector $D_q$. We try to derive a representation $u_q$ (such as a feature vector) which can be used in a way similar to a row of $U_K$ in the comparison formulas given above. One constraint is that when we input a real document $D_i$, it should give the representation $u_i$ (at least when the model is perfect, i.e. when $\hat{A} = A$). The appropriate representation of $u_q$ under such a
constraint is:

\[ u_q = D_q^T V_K \Sigma_K^{-1}. \]

This is equivalent to placing the pseudo-document at the centroid of the vectors that represent its terms (with appropriate rescaling of the axes). We can then treat \( u_q \) as a row of \( U_K \) and appropriately scale it by \( \Sigma_K^{1/2} \) in order to compare it with a term that appeared in the original analysis, or rescale by \( \Sigma_K \) to compare it with a document (just as we did above in (1) and (3)). Distances can now be computed between \( u_q \) and the documents in the corpus. In the case of query matching, the process will return the documents that most closely match the query.

### 2.3.5 Advantages and Limitations of LSA

The heuristic behind LSA is that randomness in the choice of words introduces noise into the measurement of similarity. By mapping the set of documents to a space of reduced dimensionality, called the \textit{"latent semantic space"}, some of this noise is eliminated and the strongest relationships are enhanced. Even when the original Document-Term Matrix is sparse, the resulting matrix, after reducing dimensionality, will likely not be, and the process should map synonyms to approximately the same direction in the latent space. Hence, we can measure relationships between documents even when they don’t have any common terms, due to other relationships that are present in the corpus. Studies have shown that relationships between words detected by LSA are similar to perceptions by humans using various synonym tests (Landauer et al, 1998).

Of course, LSA solves only part of the problem with polysemy. It accounts for multiple meanings of words because the meaning of a word can be described by other words in the document and by words in the query that were not used in any of the documents. However, a downfall is that each word is represented by only one point in the latent semantic space. Thus a word with multiple meanings is represented as a weighted average of its different meanings. So if none of the real meanings is similar to the average meaning, the results can be distorted.

Another essential point is that LSA does not make use of the order of the words in the documents
and therefore lacks any understanding of syntax or logical information. We propose the use of shingles of up to a fixed length instead of just single words in connection with LSA, since patterns of words are often important for detecting similarities between documents. A method alternative to LSA that also uses shingles to measure similarity between documents is proposed in Section 2.5.

2.4 Some Ideas Relating to Singular Value Decomposition

In this section, we explore two ideas connected with SVD. One is a method for creating certain new types of distance measures based on the feature vectors resulting from an SVD that may be useful for some sequencing purposes. Another is to apply the LSA method using just incidence counts (rather than frequency counts) in the Document-Term matrix. Then, we consider a logistic model to fit the probability of the occurrence of a particular term in a particular document using feature vectors as predictors. Our aim is to find the components of the document and term feature vectors that will allow us to best model the 0/1 entries of the matrix.

2.4.1 Searching for Distances that Align with Time

Many types of distances between documents can be defined, but some may not be effective for estimating the order of the documents or be relevant with respect to time separation. We will consider quadratically-based distances (between documents $i$ and $j$) of the form:

$$d(i, j) = (u_i - u_j)B(u_i - u_j)^T;$$

where $u_i$ is the feature vector for document $i$ (the $u_i$ are considered row vectors) and $B$ is a $K \times K$ non-negative definite matrix.

The goal is to find the non-negative definite matrix $B$ that makes this distance measure most aligned with time, i.e., we seek distances that will be useful for sequencing documents with respect to time using the distance given above.
The form of the distance may be written as:

$$d(i, j) = (u_i - u_j)B(u_i - u_j)^T = \sum_{r=1}^{K} \sum_{s=1}^{K} (u_{ir} - u_{jr})B_{rs}(u_{is} - u_{js})$$

and in the case where $B$ is diagonal, it simplifies to:

$$d(i, j) = \sum_{r=1}^{K} B_{rr}(u_{ir} - u_{jr})^2.$$ 

This leads us to seek distances of the form $d(i, j) = \sum_{r=1}^{K} w_r(u_{ir} - u_{jr})^2$ which assign a weight to each feature of the corpus and to attempt to estimate these weights by the following linear regression:

$$t_{ij} = \sum_{r=1}^{K} w_r(u_{ir} - u_{jr})^2 + e_{ij};$$

where $t_{ij}$ is the time difference between the dates in which document $i$ and $j$ are written, the predictors are the $(u_{ir} - u_{jr})^2$ where $u_{ir}$ represents the $r$th entry of the feature vector for document $i$, and the $e_{ij}$ are presumed to be iid $\sim N(0, \sigma^2)$ as typically done in regression work.

In practice we will not know the dates of all of our documents, however this regression procedure can potentially be useful when we have some documents with known dates. Consider the corpus $D = \{D_1, D_2, \ldots, D_n\} = \{D_1, \ldots, D_T, D_{T+1}, \ldots, D_n\} = (T, V)$ divided into a training set $T$ of $T$ documents with known dates and a test set $V$ of $n - T$ documents with unknown dates. Using the training set, we can attempt to fit weights that best predict time differences, and then calculate the resulting distances (using the fitted weights) between the documents in the test set. After obtaining these distances, the order of the documents in the test set would be estimated using the sequencing methods of Chapter 3. One would expect the estimation of ordering to be better since we are using distances that have been aligned with time rather than arbitrary distance measures that may not equally reflect the time difference between documents.
The method may be summarized as follows. First using the entire corpus $D$, we perform Latent Semantic Analysis which gives feature vectors $u_i$ of length $K$ for all the documents. Now for each (unordered) pair of documents $(i, j)$ in the training set, we fit the regression model:

$$t_{ij} = \sum_{r=1}^{K} w_r (u_{ir} - u_{jr})^2 + e_{ij}; \text{ for } i < j, i = 1, 2, \ldots, T - 1, j = 2, 3, \ldots, T$$

where $t_{ij}$ is the time between the dates in which document $i$ and $j$ are written, $u_{ir}$ is the $r$th entry of the feature vector for document $i$, and $e_{ij}$ iid $\sim N(0, \sigma^2)$. There are $K$ parameters to be estimated, corresponding to the weights of each feature of the entire corpus. The fitted weights from the regression model represent the weights of each feature of the entire corpus and should allow for distances that are relevant to time ordering. Distances between each pair of documents in the test set are then calculated using the estimated weights $\hat{w}_r$ from the regression as follows:

$$d(i, j) = \sum_{r=1}^{K} \hat{w}_r (u_{ir} - u_{jr})^2; \text{ for } i < j, i = T + 1, T + 2, \ldots, n - 1, j = T + 2, T + 3, \ldots, n$$

These distances are then used for sequencing the documents in the test set by applying the methods described in Chapter 3.

We remark here that although this method seemed promising, when tested on the DEEDS data, we found that the fitted distances between documents in the training set had a very high correlation with their true time differences, whereas the distances between documents in the test set had much lower correlations. This phenomenon most likely is the result of a degrees of freedom problem; we do not actually have the apparent number, $\binom{n-T}{2}$, of independent variables in the regression on the test set; in particular, there is dependence among all case pairs which share a common document.
2.4.2 A Logistic SVD Model

We may consider a variation of the LSA technique which involves looking at incidence counts rather than the number of times words appear in each document. One could argue that the number of times a word appears in a document is not as relevant as the fact that the word either appeared or did not appear in the document. In this case, the Document-Term matrix, \( A \) is taken to be an incidence matrix with entries \( A_{ij} = I(w_j \in D_i) \). The SVD is again written as \( A = U\ell \Sigma\ell V_\ell^T \). Letting \( U = U\ell \Sigma_\ell^{1/2} \) and \( V = V\ell \Sigma_\ell^{1/2} \), the SVD form is equivalent to \( A = UV^T \). So, the SVD attempts to write each entry \( A_{ij} = u_i v_j^T \), where \( u_i \) and \( v_j \) are row vectors of \( U \) and \( V \) respectively.

We next let \( p_{ij} = P(A_{ij} = 1) = P(w_j \in D_i) \) denote the probability that word \( j \) occurs in document \( i \). The Logistic SVD Model is defined as:

\[
\log \frac{\hat{p}_{ij}}{1 - \hat{p}_{ij}} = u_i v_j^T = \sum_{r=1}^{K} u_{ir} v_{jr} \quad \text{for } i = 1, 2, \ldots, n \text{ and } j = 1, 2, \ldots, m
\]

where the probabilities \( \hat{p}_{ij} \) as well as the \( u \)'s and \( v \)'s are to be fitted. The aim here is to find the components of the document and term feature vectors, namely the \( u_{ir} \) and \( v_{jr} \), that will allow us to model incidence counts in the Document-Term Matrix. The SVD-like fitting procedure here is not as simple as in the regular case, but it can be fit feature by feature.

Next, we introduce an entirely different approach to defining distances based on shingles.

2.5 The “Tableaux” Method

In this section, we propose a way of representing the corpus by examining each pair of documents not only individually, but also in the context of the corpus as a whole, and also accounting for various aspects of the pair of documents relating to the length of the documents and the occurrence of common substrings of assorted lengths. We use these aspects to define a measure of similarity between pairs of documents, in the context of the entire corpus.
For a given pair of documents, a tableau/chart is created via the following steps:

1. Parse both documents by removing punctuation and capitalization.

2. Find the longest common shingle between the 2 documents and record its count as the number of times it appeared commonly in both documents (i.e., the minimum between the number of times it appeared in Document 1 and in Document 2). Then remove this shingle from both documents.

3. Find the next longest common shingle and repeat the above step until there are no more common shingles remaining.

4. For each common shingle, we record the shingle name, shingle length, its count (number of times it appeared commonly in both documents), and the proportion of times the shingle appeared in the entire corpus.

5. The tableau is ordered first by shingle length, and secondarily by rarest to the most common shingles in the pair of documents, and then alphabetically.

We repeat this procedure for each pair of documents resulting in \( \binom{n}{2} \) tableaux in all.

Note that in step 2. above, we delete the common shingle, so we do not subsequently count common substrings that are a part of the longer common substring. The only way in which we would later count any substring of a previous common shingle is if it appears in different parts of the documents that are also not substrings of a shingle already accounted for. The above process can therefore produce tableaux that are not quite unique, as for example if there were a tie for the longest common shingle. (If at any stage we have two or more longest common shingles of equal length, then the tableaux may depend on which one is removed first.) We may consider a different method which will produce unique tableaux by following the above procedure, but without deleting shingles as they are used. This means that for each shingle of length \( \ell \), its substrings of sizes 1, 2, \ldots, \( \ell - 1 \) will then also be listed in the tableau, resulting in longer but unique tableaux. In either case, we can produce a tableau for each pair of documents. However, we will not use the longer of these methods.
For each pair of documents $i$ and $j$, the following quantities are recorded in the tableau:

- $n_i =$ length (number of words) of document $i$
- $n_j =$ length (number of words) of document $j$
- $m_{ij} =$ number of shingles common to documents $i$ and $j$
  (this is the number of shingles that are deleted from both documents if following the first procedure, or the total number of shingles including substrings of shingles that were already counted if following the second procedure.)
- $\ell_s =$ length of shingle $s$, $s = 1, 2, \ldots m_{ij}$
- $N_s(i,j) =$ number of times shingle $s$ occurs commonly in documents $i$ and $j$
- $p_s =$ proportion of documents in the entire corpus that contain shingle $s$

We illustrate the Tableaux method using a simple example in Appendix [A]

### 2.5.1 Tableaux-based Distances and Similarities

Tableaux are intended for defining distances or similarities. They allow us to create new and more in-depth pairwise distance measures between documents that attempt to take the entire corpus into account. One such distance is:

\[
\text{dist}(i,j) = 1 - \frac{2 \sum_{s=1}^{m_{ij}} \ell_s N_s(i,j)}{(n_i + n_j)}. \tag{2.5.1}
\]

However, two documents that are identical (same ordering of words except for capitalization, punctuation, etc.) should have distance 0. In that case, the proportion of documents containing that (long) shingle should not add more importance to the distance and it will not make sense to weight the distance with the proportion. One may consider an alternate distance which does not use the proportion of documents in the calculation:

\[
\text{dist}(i,j) = 1 - 2 \sum_{s=1}^{m_{ij}} \ell_s N_s(i,j). \tag{2.5.2}
\]
Different types of distances can also be constructed. However, defining similarity is easier using the quantities displayed in the tableaux. A similarity between documents \( i \) and \( j \) may be defined as follows:

\[
\text{sim}(i, j) = \frac{1}{(n_i n_j)\gamma} \sum_{s=1}^{m_{ij}} \frac{\ell^s_{ij}}{p^s_{ij}} N_s(i, j) ; \text{ where } \alpha, \beta, \gamma > 0. \tag{2.5.3}
\]

This seems to be a natural way to define similarity as one would expect that longer and more rarely occurring common shingles should increase similarity. We also downweight by the document length as longer documents will obviously have a higher chance to contain more common shingles. Each set of values for the parameters \( \alpha, \beta, \gamma \) will give rise to a different similarity measure. These parameter values could then be chosen empirically (typical values are \( \alpha, \beta, \gamma \in \{0.5, 1, 2\} \)). We will discuss this procedure in the next section.

### 2.5.2 Searching for Optimal Values of \( \alpha, \beta, \gamma \)

We are again interested in similarity measures correlated with separation in time. This can be done if we have a training set of documents (with known dates). Suppose we have a training set, \( T \), of size \( T \) and a test set, \( V \), of \( n \) documents with unknown dates. Typically, \( T > n \).

For each pair of documents \((i, j) \in T\) we then have:
- \( t_{ij} = \text{time between dates for document } i \text{ and } j \)
- \( \text{sim}(i, j) \) for a given set of values \((\alpha, \beta, \gamma)\).

The procedure is then as follows. We take different subsets of \( n \) documents each from the training set. For each such subset, the similarity measure is computed for all possible values of the \( \alpha, \beta, \gamma \) parameters. We then minimize the Spearman Rank Correlation between the time differences and similarity measures over the \((\alpha, \beta, \gamma)\) values for all subsets of the training set. In this case, we minimize correlation because we expect higher similarities for smaller time separations. The values of \((\alpha, \beta, \gamma)\) that minimize the correlation are considered to be the optimal parameter values.
As an example, the general results that were found when tested on the DEEDS data set are:

- For all sizes $n$, we found that $\gamma_{opt} = 0.5$.
- Optimal values vary for $\alpha$ and $\beta$ depending on $n$.

For $n = 15$ to $50$, the optimal values for the pair $(\alpha_{opt}, \beta_{opt})$ are $(1, 2), (2, 1), (1, 0.5)$, and $(2, 0.5)$ respectively.

Once the optimal parameters are found, these values are used in Formula (2.5.3) to compute similarities between all pairs of the $n$ documents in the test set $\mathcal{V}$. In cases where a training set is not available, one can use a corpus with known dates (similar to the corpus that needs to be sequenced) to try to estimate suitable parameter values.

These tableaux-based similarities can then be converted into distance measures by the methods described in the next section.

### 2.6 Converting Similarities to Distances

Converting a distance measure into a similarity can be done easily, however the reverse is much more difficult, particularly if we want the resulting distance to be a true metric. Note, however, that for the problem of sequencing elements, it is not necessary for the distances to satisfy the triangle inequality (see Section 4.2). This is in keeping with well known and general findings in most other text processing work.

To convert a given similarity measure into a distance, we first normalize the similarity measure $\text{sim}(i, j)$ so that the measure maps into $[0, 1]$. For example, we may take:

$$s_{ij} = \frac{\text{sim}(i, j) - \min(S)}{\max(S) - \min(S)}.$$

Here, $S$ consists of all pairwise (un-normalized) similarities $\text{sim}(i, j)$ and $s_{ij}$ is the normalized
similarity measure between the $i$th and $j$th document or element. A measure of $s = 0$ is interpreted as $i$ and $j$ being completely different from each other, whereas a measure of 1 means they are identical.

Given a similarity measure, such that $0 \leq s_{ij} \leq 1$, we can convert it to a distance $d_{ij}$ using one of the following methods:

(1) $d_{ij} = 1 - s_{ij}$, or more generally $d_{ij} = c - s_{ij}$ for some constant $c > 0$.

(2) $d_{ij} = \sqrt{1 - s_{ij}}$ : this will be the Euclidean distance (i.e. $n$ points can be embedded in Euclidean space such that $d_{ij}$ is the Euclidean distance between the $i$th and $j$th points) as long as the matrix containing all pairwise similarities $s_{ij}$, is positive semi-definite.

(3) $d_{ij} = -\ln(s_{ij})$.

Further details of the properties of these transformations and on converting similarities to distances can be found in Digby & Gower (1981), Gower & Legendre (1986), and Cox & Cox (2001).

In this chapter we have outlined various types of distance and similarity measures that are useful for comparing documents. Specifically, we summarized three different ways of obtaining distances between documents:

1. Represent each document as a vector as shown in Section 1.1 and then take any of the various distances/similarities from Section 2.1 between these vectors.

2. Represent the corpus with a Document-Term matrix. Then, use LSA to find document feature vectors and take various distances/similarities from Section 2.1 between these feature vectors.

3. Create a tableau for each pair of documents in the corpus. For a given set of values $(\alpha, \beta, \gamma)$, compute similarities as given in Formula (2.5.3) and then convert to distances.

For 1., many types of distances can be used, although cosine and Euclidean are the most common. For 2., the distance measure will depend on the choice of the dimensionality $K$ of
the SVD as well as the type of distance used. Again, Euclidean and cosine measures here are most common. For 3., each set of the parameters \((\alpha, \beta, \gamma)\) will give a different similarity measure. However, finding the optimal parameter values as outlined in Section 2.5.2 helps narrow down the choice of measures.

We observe that the Tableaux method described in this chapter does not appear to have been used in this specific manner in other works. Distances using shingles (such as that of Broder, 1997) have been proposed but do not use all shingle lengths at a time and consider different aspects of the corpus as we have done. Our Tableaux method is useful because it gives a way to represent the corpus and to measure distances between document pairs by using the features of the entire corpus and repeating word patterns, rather than single words like other methods such as LSA.

Now that distance measures have been determined, the next step involves ordering the documents based on those distances. The next chapter discusses the general topic of sequencing elements based on distances.
Chapter 3

Sequencing Elements via Least Squares Criteria

In this chapter, we describe a framework for sequencing a set of elements based on minimization of certain Least Squares criteria. These methods can be applied to any type of “points” or elements (not only to documents) provided that estimated pairwise distances between the elements can be obtained.

In the first section of this chapter, we introduce a basic idea behind sequencing elements based on distances, and we then suggest two regression-type models for fitting these distances in Sections 3.2 and 3.3. The mathematical patterns for the matrices and estimators arising in these regression models are interesting in their own right, and are given in Section 3.4. In Section 3.5, we discuss some properties of the Error Sum of Squares arising in these regression models. Sections 3.7–3.9 consider the use of a constrained regression estimation method, and more complex models for distances, that could be applied in sequencing problems. Finally, we propose a “Closest Peers” method for sequencing elements that is not based on a regression model in the last section 3.10.
3.1 Sequencing Elements based on Distances

Our context is that there are \( n \) elements of unknown order, and a matrix of pairwise distances between the elements where these distances have been measured with error. The goal is to optimally estimate an ordering of the elements based on the matrix of these pairwise distance measures. We thus have:

- \( \binom{n}{2} \) measurements of distance between the pairs of elements
- \( n!/2 \) possible orderings of the elements (reversal being indistinguishable).

We propose to estimate the ordering by minimizing an overall measure of distance between the elements. This can be done, for instance, by minimizing the estimated spacings, i.e., the distances between consecutive elements in the estimated ordering. This problem is in some respects similar to the “Travelling Salesman Problem”, except that here we allow for error in the distance measures. We shall propose and then examine a certain regression model on the distances for estimating the ordering.

3.2 A Linear Regression Model for Distances

A basic linear regression model for the pairwise distances is suggested in this section. Let \( X_{i,j} \) represent the available (estimated) distance between the \( ith \) and \( jth \) elements. Since these distances are measured with error, we consider the following model:

\[
X_{i,j} = d_{i,j} + e_{i,j}; \quad (3.2.1)
\]

where \( i = 1, 2, \ldots, n-1, j = 2, \ldots, n-1, n, \) and \( i < j \). Here, the \( d_{i,j} \) are meant to represent the true distances between the \( ith \) and \( jth \) elements and may be considered as being parameters to be estimated from the data. Furthermore, we will model the errors as \( e_{i,j} \sim \text{iid } N(0, \sigma^2) \), as this
choice simplifies the computations and is known to be more or less robust in typical regression-type problems. In total, there are \( \binom{n}{2} \) observations \( X_{i,j} \) corresponding to the \( \binom{n}{2} \) element pairs.

**How many parameters need to be estimated?**

At first glance it appears as if there are \( \binom{n}{2} \) parameters (namely the \( d_{i,j} \)) to be estimated. However, provided that the order of the \( n \) elements is known, then there are only \( n - 1 \) parameters that need to be estimated. This is because the distances are viewed as being metrics on \( \mathbb{R}^1 \), so that it suffices to know the distances (spacings) between the pairs of consecutive elements. These spacings would allow us to determine the distance between any pair of elements. A revised model based on such adjacent spacings is proposed in the next section.

### 3.3 A Regression Model based on Spacings

Suppose, without loss of generality, that the indices \((1, 2, \ldots, n)\) represent the true order of the elements. Then the model may be viewed as:

\[
Y_{i,i} = 0 \quad \text{for } i = 1, \ldots, n \\
Y_{i,j} = \beta_{i,i+1} + \beta_{i+1,i+2} + \ldots + \beta_{j-2,j-1} + \beta_{j-1,j} + e_{i,j} ; \quad (3.3.1)
\]

where \( i = 1, 2, \ldots, n-1, j = 2, \ldots, n-1, n, \) and \( i < j \). Here, \( Y_{i,j} \) are the observed distances between the \( ith \) and \( jth \) elements in the true ordering, \( \beta_{i,i+1} \) represents the spacing between the consecutive elements \( i \) and \( i+1 \) in this ordering, and the errors are again taken as \( e_{i,j} \sim \text{iid } N(0, \sigma^2) \). In this model, there are only \( n - 1 \) parameters, corresponding to the spacings, to be estimated.

Let \( r = \binom{n}{2} \). In matrix form, Model (3.3.1) becomes: \( \tilde{Y} = X\tilde{\beta} + \tilde{e} \). The explicit form of this
equation is shown in display 3.3.2.

Here the dependent variable $Y$ has the dimension $r \times 1$, the design matrix $X$ is $r \times (n - 1)$, the parameter vector $\beta$ is $(n - 1) \times 1$, and the error vector $e$ has dimension $r \times 1$. The error vector is modeled as $e \sim N(0, \sigma^2 I)$, where $I$ is the $r \times r$ identity matrix.

Note that for each proposed ordering of the $n$ elements, the arrangement of the distances $Y_{i,j}$ comprising the response vector, $Y$, will change. This will lead to different estimates of the $\beta$ parameters and of $\sigma^2$ for each ordering of the elements. However, regardless of the selected order for the elements, the design matrix, $X$, remains unchanged and depends only on $n$. In the next section, we develop the explicit forms of the various matrices and estimators which are associated with this regression problem.
CHAPTER 3: SEQUENCING ELEMENTS VIA LEAST SQUARES CRITERIA

3.4 Form of the Regression Matrices and Estimators

The fundamental regression matrices $X$, $X'X$, $(X'X)^{-1}$, $H = X(X'X)^{-1}X'$, and $I - H$ all depend only on $n$ and have the following interesting patterns.

The Design Matrix, $X$:

As has been noted, the Design Matrix $X$ remains unchanged regardless of the order of the elements. Each row of $X$ corresponds to a distance between two elements and contains a sequence of 1s corresponding to the adjacent spacings ($\beta_{i,i+1}$ terms) that make up that particular distance. The remaining entries are zeroes. The general form of $X$ is:

$$
X = \begin{bmatrix}
1 & 0 & 0 & 0 & \ldots & 0 \\
1 & 1 & 0 & 0 & \ldots & 0 \\
1 & 1 & 1 & 0 & \ldots & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
1 & 1 & 1 & 1 & \ldots & 1 \\
0 & 1 & 0 & 0 & \ldots & 0 \\
0 & 1 & 1 & 0 & \ldots & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 1 & 0 & \ldots & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 1 & 1 & \ldots & 1 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & 0 & \ldots & 0 \\
0 & 0 & 0 & 1 & \ldots & 0 \\
0 & 0 & 0 & 0 & \ldots & 1 \\
0 & 0 & 0 & 0 & \ldots & 0 \\
\end{bmatrix}
$$
The symmetric matrices $X'X$ and $(X'X)^{-1}$ derived from this matrix $X$ take on the following patterns. Firstly, $X'X$ is an $(n - 1) \times (n - 1)$ matrix. It consists of the numbers $1, 2, \ldots, n - 1$ on the outer rims, with the numbers $1, 2, \ldots, n - 1$ running down the last column and across the last row, and running up the first column and backwards across the first row. The entries up to and including the diagonal entry in each row increment by an amount equal to the first number in that particular row. Thus, the lower triangle of the $X'X$ matrix has the entry $n - 1$ in the first row, entries $n - 2, 2(n - 2)$ in the second row, and so on, with $2, 4, 6, \ldots, 2(n - 3), 2(n - 2)$ in the second to last row, and of course $1, 2, \ldots, n - 1$ in the last row. The remaining entries are filled in in such a way as to make the matrix symmetric. The general pattern is:

$$X'X = \begin{pmatrix}
  n-1 & n-2 & n-3 & \ldots & 3 & 2 & 1 \\
  n-2 & 2(n-2) & 2(n-3) & \ldots & 6 & 4 & 2 \\
  n-3 & 2(n-3) & 3(n-3) & \ldots & 9 & 6 & 3 \\
  \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
  3 & 6 & 9 & \ldots & 3(n-3) & 2(n-3) & n-3 \\
  2 & 4 & 6 & \ldots & 2(n-3) & 2(n-2) & n-2 \\
  1 & 2 & 3 & \ldots & n-3 & n-2 & n-1
\end{pmatrix}.$$ 

For example, for the respective cases when $n = 5$ and $n = 6$, $X'X$ is given by:

$$\begin{pmatrix}
  4 & 3 & 2 & 1 \\
  3 & 6 & 4 & 2 \\
  2 & 4 & 6 & 3 \\
  1 & 2 & 3 & 4
\end{pmatrix} \quad \begin{pmatrix}
  5 & 4 & 3 & 2 & 1 \\
  4 & 8 & 6 & 4 & 2 \\
  3 & 6 & 9 & 6 & 3 \\
  2 & 4 & 6 & 8 & 4 \\
  1 & 2 & 3 & 4 & 5
\end{pmatrix}.$$ 

Next, the matrix $(X'X)^{-1}$ contains an entry of $(n - 1)/r \equiv 2/n$ on each position of its main diagonal, an entry of $-1/n$ on each position of its two main sub-diagonals, and zeroes everywhere else. The tri-diagonal form is:
\( \begin{pmatrix} 
\frac{2}{n} & -\frac{1}{n} & 0 & 0 & \ldots & 0 & 0 \\
-\frac{1}{n} & \frac{2}{n} & -\frac{1}{n} & 0 & 0 & \ldots & 0 \\
0 & -\frac{1}{n} & \frac{2}{n} & -\frac{1}{n} & 0 & \ldots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
0 & \ldots & 0 & -\frac{1}{n} & \frac{2}{n} & -\frac{1}{n} & 0 \\
0 & 0 & \ldots & 0 & -\frac{1}{n} & \frac{2}{n} & -\frac{1}{n} \\
0 & 0 & \ldots & 0 & -\frac{1}{n} & \frac{2}{n} & \frac{2}{n} \\
0 & 0 & \ldots & 0 & -\frac{1}{n} & \frac{2}{n} & -\frac{1}{n} \\
\end{pmatrix} \)

\( (X'X)^{-1} = \begin{pmatrix} 
2 & -1 & 0 & 0 & \ldots & 0 & 0 \\
-1 & 2 & -1 & 0 & 0 & \ldots & 0 \\
0 & -1 & 2 & -1 & 0 & \ldots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
0 & \ldots & 0 & -1 & 2 & -1 & 0 \\
0 & 0 & \ldots & 0 & -1 & 2 & -1 \\
0 & 0 & \ldots & 0 & -1 & 2 & \frac{2}{n} \\
\end{pmatrix} \)

\( = \frac{1}{n} \begin{pmatrix} 
2 & -1 & 0 & 0 & \ldots & 0 & 0 \\
-1 & 2 & -1 & 0 & 0 & \ldots & 0 \\
0 & -1 & 2 & -1 & 0 & \ldots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
0 & \ldots & 0 & -1 & 2 & -1 & 0 \\
0 & 0 & \ldots & 0 & -1 & 2 & -1 \\
0 & 0 & \ldots & 0 & -1 & 2 & \frac{2}{n} \\
\end{pmatrix} \)

**The Hat Matrix, H:**

The usual Hat matrix, which is of dimension \( r \times r \) is defined as \( H = X(X'X)^{-1}X' \). Each row as well as each column of \( H \) corresponds to (a distance between) a particular pair of elements. Each entry of the Hat matrix represents the influence of each observed pairwise distance, \( Y_{i,j} \) on each fitted value of \( Y_{k,\ell} \), for \( i < j \) and \( k < \ell \). This makes the explicit form of the matrix somewhat difficult to write out although it can actually be defined very easily in terms of the indices \( i, j, k, \ell \).

In the representation shown below, \( i, j \) are the indices of the element pair corresponding to the row
and $k, \ell$ are the indices of the element pair corresponding to the column:

$$H[(i,j),(k,\ell)] = \begin{cases}
\frac{2}{n}, & i = k, j = \ell \\
\frac{2i-1}{n}, & i = k, \ell > j \\
\frac{1}{n}, & j = \ell, k > i \\
\frac{j-k-1}{n}, & i \leq k \leq j \\
0, & j < k.
\end{cases}$$

$H$ is a large $r \times r$ matrix with an entry of $2/n$ at each position on the main diagonal; it therefore has a trace of $(n-1)$. It is symmetric and idempotent ($H^2 = H$) as per the usual properties in linear regression. The matrix $H$ can be easily formed numerically, for any specific value of $n$, using an appropriate ‘R’ program, for instance. The $n = 4$ case which results in a $6 \times 6$ Hat matrix is illustrated below:

$$H = \begin{pmatrix}
0.50 & 0.25 & 0.25 & -0.25 & -0.25 & 0 \\
0.25 & 0.50 & 0.25 & 0.25 & 0 & -0.25 \\
0.25 & 0.25 & 0.50 & 0 & 0.25 & 0.25 \\
-0.25 & 0.25 & 0 & 0.50 & 0.25 & -0.25 \\
-0.25 & 0 & 0.25 & 0.25 & 0.50 & 0.25 \\
0 & -0.25 & 0.25 & -0.25 & 0.25 & 0.50
\end{pmatrix}.$$

**Properties of the Estimators:**

Finally, we turn to some properties of the estimators resulting from this regression model. It is an interesting and perhaps counterintuitive property that under our model all of the estimated spacings (the $\hat{\beta}$s) actually have identical variances. In addition, all of the fitted values for the distances, $\hat{Y}_{i,j}$, also have identical variances. One might have expected that fitted distances corresponding to a pair of elements that are far away from each other would have larger variances than that of elements
closer together, however that is not the case. Furthermore, the estimated spacings between pairs of elements that do not share any common elements (indices) are uncorrelated. The remaining covariances between spacings are all negative and are equal to half the value of the variances. The explicit forms will be shown further below.

The variance-covariance matrix of the regression coefficients is given by:

$$\hat{\text{Var}}(\hat{\beta}) = \text{MSE} \cdot (X'X)^{-1}; \text{ where}$$

- $\text{MSE} = \frac{\text{SSE}}{df} = \left(\frac{n-1}{2}\right)^{-1} Y' (I - H) Y$ is the Mean Square Error.
- $\text{SSE} = Y' (I - H) Y$ is the Error Sum of Squares for the regression model.
- The number of degrees of freedom is calculated as:

$$df = \text{the number of data points} - \text{number of parameters to be estimated} = r - (n - 1) = \left(\frac{n-1}{2}\right).$$

Since the diagonal elements of $(X'X)^{-1}$ are identical, the estimated variances of all coefficients will be the same, namely,

$$\hat{\text{Var}}(\hat{\beta}_{i,i+1}) = \frac{2}{n} \text{MSE},$$

and due to the pattern of $(X'X)^{-1}$, the estimated covariances are given by:

$$\hat{\text{Cov}}(\hat{\beta}_{i,i+1}, \hat{\beta}_{k,k+1}) = \begin{cases} \frac{2}{n} \text{MSE}, & i = k \\ -\frac{1}{n} \text{MSE}, & |i - k| = 1 \\ 0, & |i - k| \geq 2. \end{cases}$$

The tri-diagonal form of the $(n - 1) \times (n - 1)$ covariance matrix for the coefficients is:
The variance-covariance matrix for the fitted distances is given by standard computations:

\[
\widehat{\text{Var}}(\hat{Y}) = \text{Var}(X\hat{\beta}) = X[\text{Var}(\hat{\beta})]X' = \text{MSE}[X(X'X)^{-1}X'] = (\text{MSE})H.
\]

Hence, the fitted distances all have the same variance, \(\text{Var}(\hat{Y}_{i,j}) = \frac{2}{n}\text{MSE}\), since the variances are determined by the diagonal elements of \(H\).

The covariances among the fitted values have a more complicated pattern as these depend on whether the pair of distances are overlapping and by how much, or if they are disjoint (do not correspond to any elements in common). For any \(i < j\) and \(k < \ell\):

\[
\widehat{\text{Cov}}(\hat{Y}_{i,j}, \hat{Y}_{k,\ell}) = \begin{cases}
\frac{2}{n}\text{MSE} & , \ i = k, \ j = \ell \\
\frac{2i-1}{n}\text{MSE} & , \ i = k, \ \ell > j \\
\frac{1}{n}\text{MSE} & , \ j = \ell, k > i \\
\frac{i-k-1}{n}\text{MSE} & , \ i \leq k \leq j \\
0 & , \ j < k.
\end{cases}
\]
3.5 Properties of the Sum of Squared Errors (SSE)

Implicitly, the permutation order (to within reversal) of the elements is a parameter in model (3.3.1). When the elements are not placed in their correct order, the model is wrong. The following are key properties of the Error Sum of Squares which were useful in motivating how to estimate the ordering based on distances.

We begin by noting that the Error Sum of Squares for the model depends on the order of the elements. When the elements are in a given order \( \tau \), the *Error Sum of Squares (SSE)* will be written as:

\[
SSE(\tau) = Y'_\tau (I - H)Y_\tau.
\] (3.5.1)

Here, \( Y_\tau \) represents the pairwise distances response vector when the elements are arranged to correspond to the order \( \tau \). The properties of the SSE will depend on the ordering and the error variability.

**Properties of SSE:**

1) For the regression model with \( \sigma^2 = 0 \) (i.e., distances in \( \mathbb{R}^1 \), and measured without error) and with the elements being in their true permutation order, \( T \), clearly we will have \( SSE(T) = 0 \).

2) For an incorrect permutation, \( W \), and/or for any value of \( \sigma^2 > 0 \), \( SSE(W) \) will necessarily be \( > 0 \).

3) \( SSE \) will be the same for any two orderings that are the reverses of each other.

4) For any given ordering \( W \), the expected value of \( SSE(W) \) decomposes into two separately interpretable parts, the first being due solely to the (incorrect) ordering, and the second being due solely to error variability \( (\sigma^2) \). Specifically, \( E_{\sigma^2}[SSE(W) - SSE(T)] \) does not depend on
the value of $\sigma^2$. Thus, we have:

$$E_{\sigma^2}[SSE(W) - SSE(T)] = E_0[SSE(W) - SSE(T)] = E_0[SSE(W)] - E_0[SSE(T)] = SSE_0(W) - SSE_0(T) = SSE_0(W)$$

so that $E_{\sigma^2}(SSE(W)) = E_{\sigma^2}(SSE(T)) + SSE_0(W)$.

Note that when $\sigma^2 = 0$, the term $SSE(W)$ is a constant, and denoted here as $SSE_0(W)$, while the term $SSE_0(T) = 0$.

5) For a fixed $\sigma^2$, $E_{\sigma^2}(SSE(W))$ will be at a minimum when the elements are in the correct order, i.e, when $W = T$.

It follows also that when all of the errors are iid $N(0, \sigma^2)$, minimizing $SSE$ over all orderings of the $n$ elements will give the Maximum Likelihood Estimator for the correct ordering of the elements. So, the ordering that yields the smallest $SSE$ will be regarded here as the estimated optimal order.

One caveat of this method (as well as of most others) is that it will not be able to distinguish between the true ordering and its reverse (on account of Property 3).

Proofs:

For the model, $Y = X\beta + \epsilon$, the Ordinary Least Squares estimator of $\beta$ is the well known $\hat{\beta} = (X'X)^{-1}X'Y$. We keep in mind that $X$ and $H$ will be the same irrespective of the ordering of the elements used when fitting the regression.

**Proof of 1)**: Let $Y_T$ denote the response when the elements are in the true order $T$. Since the response is not measured with error and the elements are in the correct order, we can retrieve the exact distance between any two elements using the appropriate spacings. So we have, $Y_T = X\beta$ and from this it follows that $SSE_0(T) = 0$.

**Proof of 2)**: If the model is incorrect, at least one pair of distances will differ from its fitted value. Every such pair will contribute a positive value to the SSE and hence $SSE(W) > 0$. 

Proof of 3): This result follows because a regression SSE is invariant under permutations of the rows and furthermore is invariant under permutations of the columns of the design matrix. The two regressions problems (corresponding to two orderings that are reverses of each other) will therefore differ only in the reversal of the $\beta$ vector.

Proof of 4): Define the following:

$Y_T$ = response vector for the elements in the true order measured without error.

$Y$ = response vector for the elements in a wrong order (a permutation of the elements in $Y_T$) measured without error.

$e_T$ = error vector for the model with elements in the true order, $e_T \sim \text{iid } N(0, \sigma^2)$.

$e$ = error vector for the model with elements in a wrong order (a permutation of the elements in $e_T$), $e_i \sim \text{iid } N(0, \sigma^2)$.

$SSE_0(T) =$ SSE for the model with the elements in the true order and $\sigma^2 = 0$.

$SSE(T) = $ SSE for the model with elements in the true order and $\sigma^2 \neq 0$.

$SSE_0(W) = $ SSE for the model with elements in a wrong order and $\sigma^2 = 0$.

$SSE(W) = $ SSE for the model with elements in a wrong order and $\sigma^2 \neq 0$.

Note: For both the wrong and true ordering, $X$ and $H$ remain the same.

Also, $E(e_T) = E(e) = 0$ and $\text{Cov}(e_T) = \text{Cov}(e) = \sigma^2 I$.

Now,

\[
SSE_0(T) = Y_T'(I - H)Y_T = 0 \quad \text{(as shown in 1)}
\]

\[
SSE(T) = (Y_T + e_T)'(I - H)(Y_T + e_T)
\]

\[
SSE_0(W) = Y'(I - H)Y
\]

\[
SSE(W) = (Y + e)'(I - H)(Y + e)
\]

Next, recall the property for expectations of quadratic forms:

\[E(Z' BZ) = [E(Z)]' B [E(Z)] + \text{tr}(BC), \text{ where } Z \text{ is a random vector, } B \text{ a matrix, and } C = \]
Cov(Z). Then, we have:

\[
E_{\sigma^2}[SSE(T)] = E[(Y_T + e_T)'(I - H)(Y_T + e_T)] \\
= Y_T'(I - H)Y_T + E(e_T'(I - H)e_T) \\
= SSE_0(T) + \sigma^2(m - (n - 1)) \\
= \sigma^2(r - (n - 1))
\]

Similarly,

\[
E_{\sigma^2}[SSE(W)] = Y'(I - H)Y + E(e'(I - H)e) \\
= SSE_0(W) + \sigma^2(r - (n - 1))
\]

Therefore, \( E_{\sigma^2}[SSE(W) - SSE(T)] = SSE_0(W) \).

**Proof of 5):** This follows directly from 2) and 4). Since \( E_{\sigma^2}(SSE(W)) > E_{\sigma^2}(SSE(T)) \) for any wrong ordering, then for a fixed value of \( \sigma^2 \), \( E(SSE) \) must be minimum when the elements are in the true order.

We now proceed to estimation of the error variance, \( \sigma^2 \).

### 3.6 Concerning Estimates of the Error Variance, \( \sigma^2 \)

If the ordering of the elements is correct, then we can use the usual MSE estimate for \( \sigma^2 \) and we will have \( SSE \sim \sigma^2 \chi^2_\nu \) where \( \nu = \binom{n - 1}{2} \). However, if the ordering is incorrect, the estimate of \( \sigma^2 \) will not be reliable. If we denote the ordering that yields the smallest SSE as ‘opt’, then we could consider the estimator \( \hat{\sigma}^2 = \binom{n - 1}{2}^{-1} SSE(\text{opt}) \) since \( SSE(\text{opt}) \) is the MLE for the SSE of the true permutation. However, if we use the ordering with the minimal SSE, typically, this estimate will underestimate \( \sigma^2 \) as it is possible for different permutations other than the true one to have a smaller SSE than the true permutation. This is because the true ordering corresponds to
the ordering that minimizes the *expected* SSE, not the *actual* value of the SSE. As the number of possible permutations exceeds the degrees of freedom of the regression, the variance estimate need not be consistent.

We observe here that when \( n \) is large, it is not feasible to carry out computations for each possible permutation of the elements in order to find the one that corresponds to the minimal SSE. In such cases, we will instead use various optimization algorithms to estimate the ordering (to be discussed in Chapter 4). These optimization algorithms may therefore not necessarily arrive at the global minimum; this phenomenon thus contributes to the possible overestimation of \( \sigma^2 \). We thus have both overestimation as well as underestimation occurring for the variance parameter. We remark, however, that estimation of the variance parameter is in any case typically not the goal of the analysis. Examples and more discussion on the estimation of \( \sigma^2 \) will be given in Sections 3.8 and 4.2. In addition, an alternative method for estimating \( \sigma^2 \) using an MCMC approach is provided in Section 5.4.

### 3.7 Constrained Estimation of Spacings

In the estimation process described above, the interpoint spacings \( \beta_{i,i+1} \) were not constrained to be non-negative, although strictly speaking they should be. Hence, it is possible to get negative estimated values for some of the spacings when following the optimization procedure described above. Constrained estimation is harder to implement but could potentially provide somewhat more accurate results. Nevertheless, the unconstrained estimation should still produce reasonable results, and in particular should be able to help alert us to when an estimated ordering is far from correct, and further, may also suggest how to improve it. For instance, if an estimated ordering results in \( \hat{\beta}_{2,3} < 0 \), it suggests that the third element in the ordering should perhaps have come before the second element, although other more complicated possibilities exist, since the value of \( \hat{\beta}_{2,3} \) enters into many of the other \( \hat{Y}_{i,j} \).
3.7.1 Estimation under Non-Negativity Constraints

We consider a constrained optimization method to minimize SSE subject to non-negative spacings \( \beta_{i,i+1} \geq 0 \). The regression model for the distances using spacings is the same as Model (3.3.1):

\[
Y_{i,i} = 0 \quad \text{for } i = 1, 2, \ldots, n
\]
\[
Y_{i,j} = \sum_{k=i}^{j-1} \beta_{k,k+1} + e_{i,j} \quad \text{for } i < j, i = 1, 2, \ldots, n - 1 \text{ and } j = 2, \ldots, n - 1, n \; \text{where each } \beta_{k,k+1} \geq 0 \text{ for } k = 1, 2, \ldots, n - 1.
\]

In matrix form,
\[
\mathbf{Y} = \mathbf{X} \hat{\beta} + \mathbf{e} \quad \text{where}
\]
\( \mathbf{Y} \) is \( r \times 1 \), \( \mathbf{X} \) is \( r \times (n - 1) \), \( \hat{\beta} \) is \( (n - 1) \times 1 \), \( \mathbf{e} \) is \( r \times 1 \), and \( \mathbf{e} \sim \mathcal{N}(0, \sigma^2 \mathbf{I}) \), \( \mathbf{I} \) is \( r \times r \).

We wish to find the constrained estimator \( \hat{\beta}_{\text{con}} \), i.e. the value of \( \hat{\beta} \) that minimizes

\[
SSE_{\text{con}} = \sum_{1 \leq i < j \leq n} \left( Y_{i,j} - \sum_{k=i}^{j-1} \beta_{k,k+1} \right)^2
\]

subject to the constraints \( \beta_{k,k+1} \geq 0 \) for each \( k = 1, 2, \ldots, n - 1 \). That is,

\[
\hat{\beta}_{\text{con}} = \arg\min_{b \in [0, \infty)^{(n-1)}} (Y - Xb)'(Y - Xb).
\]

**Implementation:**

We implemented a constrained estimation algorithm using the ‘nnls’ (non-negative least squares) function in ‘R’. This function can be found in package ‘nnls’ written by Mullen and van Stokkum (2007), which provides an interface to the Lawson-Hanson algorithm for non-negative linear least squares estimation. The Lawson-Hanson algorithm solves the problem: \( \min \| Ax = b \|_2 \) under the
CHAPTER 3: SEQUENCING ELEMENTS VIA LEAST SQUARES CRITERIA

constraint \( x \geq 0 \); where \( x \in \mathbb{R}^p \), \( b \in \mathbb{R}^k \), and \( A \) is a \( k \times p \) matrix. See Lawson & Hanson (1974, 1995) for details. In our case, \( A = X \), the \( r \times (n - 1) \) design matrix; \( x = \beta \in \mathbb{R}^{n-1} \), the vector of spacings; and \( b = \tilde{Y} \in \mathbb{R}^r \), the response vector of pairwise distances.

Here again, the objective function (i.e. the Constrained Sum of Squared Errors) will be a function of the order of the elements, \( \tau \), and should therefore be written as \( SSE_{con}(\tau) \). We can compute \( SSE_{con}(\tau) \) for different permutations \( \tau \) in order to search for the one that minimizes the objective function. If ‘opt’ denotes the permutation with the minimal value for the constrained SSE, we can then estimate the error variance by \( \hat{\sigma}^2_{con} = \left( \frac{n - 1}{2} \right)^{-1} SSE_{con}(opt) \).

3.8 Sequencing using NNLS vs. Unconstrained Regression

When all estimated spacings happen to be non-negative in the unconstrained regression, both NNLS regression and unconstrained regression will give identical results. When there are negative estimated values for some \( \beta \), we have found that sequencing using NNLS regression typically provides an improvement, as expected. However, the unconstrained regression can still be useful in the sense that the negative estimated \( \beta \) values could help in suggesting which elements may have been placed out of order, and perhaps in suggesting how to improve the estimated ordering. The following example illustrates these points:

Example: Comparing Sequencing using NNLS and Unconstrained Regression

The following simulated example illustrates some of the differences between the unconstrained regression and the NNLS regression procedures for sequencing elements.

We randomly generated \( n = 4 \) points (denoted ‘numbers’) uniformly from the integers 1-100 without replacement. The observed distances between each pair of elements were calculated as the true Euclidean distance in \( \mathbb{R}^1 \) + random \( N(0, \sigma^2) \) noise. The standard deviation is denoted as ‘sigma’ in the output and in this case was chosen to have a value of 8. Since \( n \) (i.e. the number of elements) is small and hence the points are spread fairly far apart, we would expect sequencing
from both methods to be accurate. Hence, a fairly large value for $\sigma$ was selected on purpose so that there would be a sufficient error in the distances to allow us to detect a difference between the two estimation methods. The objective function ($SSE_{r}$ or $SSE_{con}(\tau)$) was calculated for each possible permutation of the elements. Both regressions use the exact same data in order to compare their accuracy in sequencing the elements. Below are the results:

Selected Numbers: 52 50 79 80

True Sigma = 8

==== SEQUENCING RESULTS for n= 4 using Unconstrained Least Squares Optimization ====

All Permutations and their SSE values:

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[,1]</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>103.13764</td>
</tr>
<tr>
<td>[,2]</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>98.19159</td>
</tr>
<tr>
<td>[,3]</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>1835.48964</td>
</tr>
<tr>
<td>[,4]</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>1608.10849</td>
</tr>
<tr>
<td>[,5]</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>2111.55721</td>
</tr>
<tr>
<td>[,6]</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1600.12543</td>
</tr>
<tr>
<td>[,7]</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>132.63023</td>
</tr>
<tr>
<td>[,8]</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>127.68418</td>
</tr>
<tr>
<td>[,9]</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>1170.13618</td>
</tr>
<tr>
<td>[,10]</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>1600.12543</td>
</tr>
<tr>
<td>[,11]</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>1470.09530</td>
</tr>
<tr>
<td>[,12]</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>1608.10849</td>
</tr>
<tr>
<td>[,13]</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1697.47645</td>
</tr>
<tr>
<td>[,14]</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>1470.09530</td>
</tr>
<tr>
<td>[,15]</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1681.56796</td>
</tr>
<tr>
<td>[,16]</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>2111.55721</td>
</tr>
</tbody>
</table>
CHAPTER 3: SEQUENCING ELEMENTS VIA LEAST SQUARES CRITERIA

[17,] 3 4 1 2 127.68418
[18,] 3 4 2 1 98.19159
[19,] 4 1 2 3 1681.56796
[20,] 4 1 3 2 1170.13618
[21,] 4 2 1 3 1697.47645
[22,] 4 2 3 1 1835.48964
[23,] 4 3 1 2 132.63023
[24,] 4 3 2 1 103.13764

Estimated Permutation 1 2 4 3 with SSE= 98.19159
Estimated Ordering 52 50 80 79 with SSE= 98.19159

Estimated Sigma= 5.72106

Estimated Betas: -6.445978 31.72938 -2.260564

============= SEQUENCING RESULTS for n= 4 using Non-Negative Least Squares Optimization =============

All Permutations and their Constrained SSE values:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[1,]</td>
<td>1 2 3 4</td>
<td>186.2389</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[2,]</td>
<td>1 2 4 3</td>
<td>191.5132</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[3,]</td>
<td>1 3 2 4</td>
<td>1835.4896</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[4,]</td>
<td>1 3 4 2</td>
<td>1608.1085</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[5,]</td>
<td>1 4 2 3</td>
<td>2111.5572</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[6,]</td>
<td>1 4 3 2</td>
<td>1623.9739</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[7,]</td>
<td>2 1 3 4</td>
<td>132.6302</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[8,]</td>
<td>2 1 4 3</td>
<td>137.9045</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[9,]</td>
<td>2 3 1 4</td>
<td>1282.2795</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Estimated Permutation 4 3 1 2 with constrained SSE= 132.6302
Estimated Ordering 80 79 52 50 with constrained SSE= 132.6302

Estimated Sigma= 6.64907

Estimated Betas: 1.623922 23.9755 5.177291

Recall that permutations that are reverses of one another have the same SSE values and are considered to be identical. Note that the negative $\hat{\beta}$ estimated values from the unconstrained regression correspond to elements that were placed out of order. For instance, $\hat{\beta}_{1,2} < 0$ and $\hat{\beta}_{3,4} < 0$ suggest that the first element in the estimated ordering should perhaps come before the second and that the third element in the estimated ordering should perhaps come before the fourth. The NNLS version has estimated the true ordering here with higher accuracy and has in fact corrected
the elements that were incorrectly ordered in the unconstrained regression. The $\beta$ estimates from NNLS are fairly close to their true values except for $\hat{\beta}_{3,4} = 5.1773$ which corresponds to the spacing between 52 and 50. This occurred due to the large error variation, and although the points 50 and 52 are actually close, their observed distance was not. Also, note that for the estimated orderings, $SSE_{\text{con}} > SSE$ as should be, and both estimates of $\sigma$ are somewhat close to the true value of the standard deviation (with the constrained estimate for $\sigma$ necessarily being somewhat higher than the unconstrained one).

In the next two sections we suggest criterion functions other than the SSE that could be useful for sequencing elements.

### 3.9 Generalizing the Criterion Function

So far, we have used SSE as the criterion function in our ordinary least squares (OLS) regression context. We now consider more complex models for distances that may also be more realistic. For example, when the errors $e_{i,j}$ are not Gaussian, have different variances, or are correlated, we can consider robust regression methods such as M-estimation instead of OLS regression. This leads to more general criterion functions:

$$\sum_{i=1, i<j}^{n} \rho(\omega_{ij}) = \sum_{i=1, i<j}^{n} \rho(y_{i,j} - x'_{i,j}\hat{\beta}).$$

The function $\rho$ defines the contribution of each residual $\omega$ to the criterion function, and should normally satisfy the following properties:

- $\rho(0) = 0$
- $\rho(\omega) \geq 0$
- $\rho(\omega) = \rho(-\omega)$
- $\rho(\omega_k) \geq \rho(\omega_{\ell})$ if $|\omega_k| > |\omega_{\ell}|$. 
For the least-squares regression, $\rho(\omega) = \omega^2$. In this general context, $\hat{\beta}$ is typically estimated by Iteratively Reweighted Least Squares (IRLS). See, for example, Huber (1964) or Fox (2002) for further details on robust estimation procedures and IRLS. Here again, the generalized criterion function will be a function of the permutation and we attempt to minimize it in order to estimate the ordering.

### 3.10 A “Closest Peers” Method for Sequencing Elements

In this section, we introduce a different method for sequencing elements which is not based on a regression on the distances as done in the previous sections. Rather, here we consider a criterion function that for a given ordering, measures the ‘closeness’ of each element to its ‘peers’. The term ‘peers’ for each element is used to describe the elements that are the most similar to that particular element. The idea is that in the true ordering of the elements, each element should be located in some sense ‘close’ to each of its peers. (i.e. in the true ordering, each document should be located on average near to the documents that are the most similar to it in content.) We call this criterion function the “Average Distance to Peers” (ADP) and define it as follows:

$$ADP(\tau) = \frac{1}{nP} \sum_{i=1}^{n} \sum_{j \in P_i} |\tau^{-1}(i) - \tau^{-1}(j)|.$$  \hspace{1cm} (3.10.1)

In the above, the set $P_i$ contains the $P$ peers of the $i$th element, where the number of peers $P$ is chosen taking into account the total number of elements, $n$. Typically, values of $P = 5, 10$ are considered. The peers of each element are identified using the distance matrix of pairwise distances between elements (and does not take the ordering, $\tau$, into account). Namely, $P_i$ will contain the $P$ elements that have the $P$ smallest values amongst all the distances in the $i$th row of the distance matrix. This ADP measure can be interpreted as the average number of steps each element must take in the given ordering to reach its $P$ most similar peers.

Nearest neighbour types of methods are commonly used in the literature, particularly for classifi-
cation problems, in which each element would be classified as belonging to the same category as its closest neighbour. However, the above ADP measure has not been proposed in this manner before and for each element, we use the closest $P$ elements rather than just one single element. Here we choose to use the terminology ‘peers’ as it indicates similarity in nature/content, whereas the term ‘neighbour’ would refer to closeness in proximity within the ordering of the elements. Thus, the Closest Peers method attempts to estimate an ordering so that on average, each element’s closest peers will be its nearest neighbours.

As before, the ADP criterion is a function of the ordering, $\tau$, and each ordering of the elements will yield a different value. We would expect the measure to be small for the true ordering. Thus, the ordering that minimizes the ADP will be considered as the estimated ordering of elements. Intuitively, this criterion function is sensible for sequencing documents since one would expect the most similar documents to be grouped together in the true ordering of the documents. Hence, rather than considering distances between all pairs of documents and imposing a regression model structure on these distances (as was done in the case of the SSE criterion), the ADP criterion focuses on closeness only to peers. For these reasons, we expect the SSE criterion function to sequence more accurately when the distances between elements are generated according to Model (3.3). On the other hand, ADP may be able to give more accurate results when applied to real document collections since the pairwise distances will not have been generated according to any regression model and will in fact be computed using word similarities between documents as described in Chapter 2. Nonetheless, both criterion functions should perform reasonably both in sequencing elements that are simulated with distances according to Model (3.3) as well as real documents. Examples comparing sequencing results for the constrained SSE and ADP criterions will be given in Section 4.3 as well as in Chapter 6. We also note that in the Closest Peers method, because the distances are not generated from any such model, we cannot estimate parameters (such as quantifying time gap between documents and there is no need to estimate $\sigma^2$, the variance of the errors).

In this chapter, we have proposed methods for how to sequence elements based on regression
models on the pairwise distances, and a closest peer method; namely minimize the SSE or ADP or appropriate objective function amongst all orderings of the elements, and the ordering with the lowest value for the objective function is taken to be the estimated optimal order. In the next chapter we will discuss approaches for efficiently calculating the objective function for different orderings. This can be done either via optimization algorithms or via MCMC methods which are the subjects of the next two chapters.
In the general problem of sequencing elements, one wishes to find an ordering that optimizes a specified criterion function. Such a criterion is a function of the ordering and can be defined in different ways, but should be invariant under order reversals as one cannot distinguish between an ordering and its reverse based solely on distances between the elements. In our case, the criterion function may be $SSE_{con}(\tau)$ as defined in Section 3.7.1, $ADP(\tau)$ as defined in Section 3.10 or some general function such as $\rho(\tau)$ as described in Section 3.9. One way to find the estimated optimal ordering is to compute the criterion function for every possible permutation of the elements as was done for the example with a small number of elements in Section 3.8. In practice, however, $n$, i.e. the number of elements to be sequenced, can be large so it will not generally be feasible to try all $n!/2$ orderings. Hence, we need to consider efficient algorithms for arriving at an estimated ordering without trying all possible permutations. Since we are searching for a particular permutation that is ‘optimal’ in some way amongst a finite set of permutations, this can be viewed as a “Combinatorial Optimization Problem”. Papadimitriou and Steiglitz (1982,1998) provide more background on combinatorial optimization.

For each permutation, the criterion function will be calculated. There are several ways one could
consider for moving from one permutation to another in an attempt to reduce the criterion function. In our experimentation, we examined a number of possible methods for permuting elements:

- Removing one element at random and placing it into a new randomly selected position (while keeping the relative order of the other elements fixed).
- Removing \(K\) elements at random and randomly inserting them into different positions.
- Removing a consecutive block of elements at random and moving that entire block to a new random position.
- Switching the position (either randomly or systematically) of pairs of elements in the permutation (Transposition).
- Switching adjacent pairs of elements in the permutation (either randomly or systematically).

It turns out that the first method above, which we will call the “Greedy Take and Put One” algorithm, proved to be the most effective for sequencing elements when tested on simulated data. Specifically, we randomly generated \(n\) numbers in \(\mathbb{R}^1\) and calculated the distances between each pair of points using different methods (such as Euclidean distances, random distances, distances that are monotone functions of the true distances) and then added random noise. This process was repeated for different values of \(n\) and for different samples of randomly selected points, with some sets containing points that are close to each other, and/or spread far apart, etc. We also considered samples of different sizes \(n\) where the points were spaced equally over the interval. In each of these cases, the ultimate estimated ordering based on the “Take and Put One” algorithm turned out to generally be better (i.e. closer to the true ordering) than that of the other methods. Also, the “Greedy Take and Put One” was faster computationally than most of the other methods. In general, we found that removing \(K\) elements at a time and reinserting them was the least effective of the methods. The “Take and Put One” method is described in detail in the next section along with examples to illustrate experimentation with different distances in Section 4.2.
4.1 “Greedy Take and Put One” Algorithm

The “Greedy Take and Put One” Algorithm moves among permutations by randomly removing one element at a time, and randomly placing it into a new position in the permutation, while keeping the relative order of the other elements fixed. It is “greedy” in the sense that it makes the optimal choice to improve the criterion value at each stage without carrying out the computations for all possible orderings. If the criterion is reduced by moving a particular element to the other place, then that element is moved and kept there, whereas if moving that element to that new position increases the criterion value, the element is not moved. The algorithm works as follows:

“Greedy Take and Put One” Algorithm:

The procedure starts by creating a \( n(n - 1) \times 2 \) ‘pairs matrix’. Each row in this pairs matrix consists of element pairs \((i, j)\), where \(i\) represents the position of the element that we will ‘take’ and \(j\) represents the new position that we will ‘put’ the selected element into. The steps below are described using the constrained SSE criterion, however, the algorithm can be applied to any appropriate criterion function as discussed earlier.

1. Start with an initial permutation. Compute the SSE [denoted as \(SSE_{con}(current)\)].

2. Randomly permute the rows of the pairs matrix. Select the first row of the pairs matrix, denoted as \((i, j)\).

3. Remove the element that is in the \(i\)th position from the current permutation and place it into the \(j\)th position (while keeping the relative order of all the other points fixed). Compute the new SSE [denoted as \(SSE_{con}(next)\)].

4. If \(SSE_{con}(next) < SSE_{con}(current)\), then keep the element in the new position and let current\(=next\). Go back to Step 2 and continue.

Else, leave the current permutation as is and select the next row in the pairs matrix, denoted as \((i, j)\). Go back to Step 3 and continue.
5. Continue the above steps until you have reached the bottom of the pairs matrix but still have not reduced the SSE of the current permutation. (i.e. stop when each of the \( n(n-1) \) different possibilities obtained by removing an element from the current permutation and placing it somewhere else has a higher SSE than the current permutation).

One iteration of the above algorithm is completed when a new permutation is accepted (i.e. the next permutation results in a lower SSE than the current permutation). Hence, in one iteration this algorithm will try at most \( n(n-1) \) possible permutations. On the last iteration before stopping, it will of course try all \( n(n-1) \) take and put one possibilities on the current permutation since the algorithm is defined in this way (as described in Step 5. above).

At the end of this procedure, we will obtain an estimated optimal ordering (denoted ‘opt’) that attempts to minimize the criterion function and has SSE denoted by \( SSE_{\text{con}}(\text{opt}) \). The error variance is then estimated by \( \hat{\sigma}^2 = \left( \frac{n-1}{2} \right)^{-1} SSE_{\text{con}}(\text{opt}) \) as suggested in Section 3.6. Although our focus is on the SSE criterion, this algorithm can be used on any objective function by replacing \( SSE_{\text{con}}(\tau) \) with \( \rho(\tau) \). Estimation of \( \sigma \) will of course vary according to the function \( \rho \).

The above algorithm will not necessarily converge to the global minimum and could result in finding only a local minimum. We remind the reader that local minima will typically occur in pairs as every ordering and its reverse have the same SSE value (on account of Property 3 in Section 3.5). For each new permutation, the criterion function is recalculated from scratch as there is no simpler way to compute the SSE from one permutation to the next. However, the design matrix stays fixed, and this simplifies and significantly speeds up the required computations. One could instead consider a non-random variant of the “Take and Put One”, in which the elements are taken systematically and put into new positions. We have chosen to use the random version because it is generally known that for different algorithms (such as in the case of MCMC and certain other optimization algorithms) randomization tends to be more efficient.

The examples of the next section illustrate sequencing results using this algorithm for distances that have been generated in different ways.
4.2 Examples using “Greedy Take and Put One” Algorithm with SSE Criterion Function

We demonstrate the sequencing results for two simulated data sets: one in which observed distances are generated from the true distances and another in which the observed distances are not generated from the true distances but rather by a monotonic function of the true distances, with random error added to the distances in both cases.

Example 1: Using True Distances

\( n = 30 \) points (labelled ‘numbers’ in the output) were randomly generated uniformly from the integers 1-100 without replacement. The observed distances are calculated as the true Euclidean distances on \( \mathbb{R}^1 \) between each pair of elements + independent random \( N(0, \sigma^2) \) noise. The error standard deviation is labelled as ‘sigma’ in the output and in this example was chosen to have a value of \( \sigma = 8 \).

The output below includes the following details:

(a) the specific 30 points that have been randomly chosen

(b) the initial random order of these 30 points as fed to the algorithm

(c) the true order, for the purpose of establishing the SSE of the true order

(d) the final estimated ordering produced by the algorithm and its corresponding SSE

(e) the number of iterations and total number of permutations tried by the algorithm before producing the final estimated ordering

(f) the true value of \( \sigma \) and its estimated value

(g) the estimated adjacent spacings and the true spacings, \( \beta s \).

The output was as follows:
(a)
Numbers (presented in true order):
4 6 14 16 17 18 22 29 30 32 34 36 40 43 45
50 55 59 62 64 68 71 72 73 74 75 78 80 94 98

====================================================================
RESULTS of Greedy TakeandPutOne for n=30 using
Non-Negative Least Squares Optimization
====================================================================

(b)
Initial Order:
45 62 29 16 18 98 34 74 14 55 36 59 68 71 17
22 50 78 64 32 73 43 72 6 80 4 40 75 94 30
SSE: 307163.5

(c)
True Order:
4 6 14 16 17 18 22 29 30 32 34 36 40 43 45
50 55 59 62 64 68 71 72 73 74 75 78 80 94 98
SSE: 23226.37

(d)
Estimated Order:
4 6 17 16 14 18 22 29 32 30 34 36 40 43 45
50 55 59 62 64 71 68 72 73 75 74 78 80 94 98
SSE: 23053.47
(e)
Number of Iterations: 88
Number of Permutations Tried: 2080

(f)
True Sigma= 8
Estimated Sigma= 7.535379

(g)
Estimated Betas:
2.3708  9.1043  2.1530  1.0549  1.6442  4.5911  5.5151  2.4118  0.6728  2.5912
1.9648  2.8498  2.5223  1.4558  7.2319  2.8391  2.7203  5.3685  3.2267  3.5418
0.1521  2.2242  1.8412  0.7110  0.6525  4.2317  2.4085 12.7881  3.3780

True Spacings:
2 8 2 1 1 4 7 1 2 2 2 4 3 2 5 5 4 3 2 4 3 1 1 1 1 3 2 14 4

The output for each of the iterations is omitted for brevity. The algorithm ran for 88 iterations and tried a total of 2080 different permutations before finally stopping. The data generation and sequencing process took less than one minute in total on a Lenovo laptop with an i7-2640M CPU at 2.80 GHz. We can see that the sequencing was done quite well. The numbers 14, 16, 17; 30, 32; and 74, 75 are slightly misplaced because they are very close to each other and because we have chosen a large variance for the errors added to the distances. Otherwise, the sequencing is perfect. Also, most of the estimated $\beta$s are close to the true spacings. The error variance is slightly underestimated corresponding to the fact that the estimated ordering happens to have a slightly smaller SSE than the true one in this instance.
Example 2: Using Monotone Distances

$n = 10$ points (labelled as ‘numbers’ in the output) were randomly generated without replacement from the integers 1-100. The observed distances between each pair of elements were assigned as follows: First, $\binom{10}{2}$ random distances between 0-99 were simulated and then sorted in ascending order. The true distances between the points on $\mathbb{R}^1$ were also calculated and sorted in ascending order. The observed distance between the closest pair of elements (the pair with the smallest true distance) was assigned the smallest random distance, the next closest pair was assigned the next smallest random distance, etc. Thus the observed distances are a haphazard monotone function of the true distances. Then a random $N(0, \sigma^2)$ noise was added to all distances as well. The error standard deviation $\sigma$ is denoted as ‘sigma’ in the output and has a true value of 5 in this example. Note that in this case, the value of $\sigma$ has no meaning since the observed distances $Y_{i,j}$ are not generated according to Model (3.3.1). However, we have included the random noise so that the distances will not be completely monotone.

The output below shows all of the same details as the previous example. In addition, it also contains the $10 \times 10$ generated distance matrix of the elements, and the correlation between the true adjacent spacings and the estimated spacings:

Numbers (presented in true order):
21 22 23 44 47 57 66 78 80 96

Distance Matrix (presented in true order):

\[
\begin{bmatrix}
21 & 22 & 23 & 44 & 47 & 57 & 66 & 78 & 80 & 96 \\
21 & 0.00 & 6.56 & 14.56 & 42.08 & 58.68 & 75.79 & 78.56 & 87.57 & 89.52 & 96.36 \\
22 & 6.56 & 0.00 & 8.26 & 34.25 & 41.30 & 50.45 & 72.44 & 71.50 & 90.65 & 107.97 \\
23 & 14.56 & 8.26 & 0.00 & 30.40 & 46.73 & 49.63 & 62.26 & 71.35 & 77.93 & 95.99 \\
44 & 42.08 & 34.25 & 30.40 & 0.00 & 8.28 & 17.08 & 33.27 & 54.28 & 61.62 & 82.47 \\
47 & 58.68 & 41.30 & 46.73 & 8.28 & 0.00 & 13.78 & 38.26 & 55.34 & 62.50 & 82.98 \\
57 & 75.79 & 50.45 & 49.63 & 17.08 & 13.78 & 0.00 & 10.73 & 28.23 & 39.15 & 63.55
\end{bmatrix}
\]
CHAPTER 4: OPTIMIZATION ALGORITHMS FOR SEQUENCING PROBLEMS

[66] 78.56  72.44  62.26  33.27  38.26  10.73  0.00  20.49  30.55  50.89
[78] 87.57  71.50  71.35  54.28  55.34  28.23  20.49  0.00  9.02  23.61
[80] 89.52  90.65  77.93  61.62  62.50  39.15  30.55  9.02  0.00  24.34
[96] 96.36  107.97  95.99  82.47  82.98  63.55  50.89  23.61  24.34  0.00

============================================================================

RESULTS of Greedy TakeandPutOne for n=10 using
Non-Negative Least Squares Optimization

============================================================================

Initial Order: 23 22 80 57 47 66 96 21 78 44
SSE: 67558.89

True Order: 21 22 23 44 47 57 66 78 80 96
SSE: 2231.87

Estimated Order: 21 22 23 44 47 57 66 78 80 96
SSE: 2231.87

Number of Iterations: 16
Number of Permutations Tried: 158

True Sigma= 5
Estimated Sigma= 7.873778

Estimated Betas:
True Spacings:
1 1 21 3 10 9 12 2 16

Correlation between true spacings and Estimated Betas: 0.9767721

The entire output for each iteration has again been omitted for brevity. The algorithm ran for 16 iterations and tried a total of 158 different permutations before stopping. The whole process took just a few seconds on a Lenovo laptop with an i7-2640M CPU at 2.80 GHz. We observe that the ordering was estimated perfectly. The estimated $\beta$s are far from their true values since the observed distances were generated from non-Euclidean distances. However, the correlation between the true and estimated spacings is high (approximately 0.98). Also, note that $\sigma$ is overestimated in this particular example even though the algorithm arrived at the true ordering. Of course in this case, estimation of $\sigma$ is meaningless, as pointed out before. Using these monotone distances, we see that even though the estimates for the $\beta$s and $\sigma$ are not meaningful, the algorithm did well with respect to sequencing the elements. We remark that this example is not an isolated one, and the results seen in this experiment are quite typical for data that have been generated in this manner.

Although the regression model (3.3.1) from Section 3.3 treats the distances as being Euclidean on $\mathbb{R}^1$, the actual distances need not be. Such examples show that the “Greedy Take and Put One” algorithm works reasonably well for other types of “distances”, as long as monotonicity holds (i.e. two elements that are closer together should, on average, have a smaller distance than two elements that are farther apart). This fact is useful because pairwise distances between documents will not be generated according to Model (3.3.1) but rather would be obtained using word similarities and the methods described in Chapter 2. For the most part, we would expect documents of a specific corpus that were written closer together in time to have smaller distances than those that were written farther apart, thus allowing for useful sequencing results.

In the next section, we will demonstrate the Greedy Take and Put One algorithm for minimizing the ADP criterion based on Closest Peers and compare it to using the SSE criterion function with NNLS for simulated data.
4.3 Comparing the SSE and ADP Criterion Functions

In this section, we present an example that compares sequencing results on simulated data using NNLS based on the SSE criterion and the Closest Peers method based on the Average Distance to Peers (ADP) criterion. \( n = 50 \) points (i.e. ‘numbers’) were randomly generated uniformly from the integers 1-100 without replacement. The observed distances are calculated as the true Euclidean distances on \( \mathbb{R}^1 \) between each pair of elements + independent random \( N(0, \sigma^2) \) noise. The error standard deviation is denoted by ‘sigma’ in the output shown below and was chosen to have a value of \( \sigma = 8 \). The sequencing results for both SSE and ADP criteria are displayed below using this same generated data. \( P = 10 \) was chosen for the number of peers for each element for the Closest Peers Method. The output shows the same details as the examples in the previous Section 4.2.

Numbers (presented in true order):
4 6 11 13 14 15 16 21 24 26 29 31 33 36 37 39 41 42 43 44 45 46 47 48 50
52 53 54 56 58 65 66 68 69 71 75 77 80 81 84 85 86 87 88 90 91 94 95 99 100
Sigma = 8

========= RESULTS of Greedy TakeandPutOne for n=50 using Closest Peers Method ==========

P = 10 Peers for each element

Initial Order: 36 50 16 99 14 26 31 46 68 33 6 91 100 66 37 58 52 77 13 69 39 45 44 54 75 53
41 29 56 44 85 47 80 24 21 48 94 95 90 4 43 84 11 86 88 71 87 81 15 65 42
ADP = 14.392

True Order: 4 6 11 13 14 15 16 21 24 26 29 31 33 36 37 39 41 42 43 44 45 46 47 48 50
52 53 54 56 58 65 66 68 69 71 75 77 80 81 84 85 86 87 88 90 91 94 95 99 100
ADP = 4.454
Estimated Order:  4  6  13  11  14  15  16  24  21  26  29  31  37  33  39  36  45  44  42  41  43  46  50  47  48  
                  53  56  52  54  58  65  68  66  69  71  75  77  81  80  86  84  88  87  85  90  95  91  99  94  100  
ADP = 4.328

Number of Iterations: 145
Number of Permutations Tried: 6172

Spearman Rank Correlation between True and Estimated Orderings: 0.9953902
Euclidean Distance between True and Estimated Orderings: 224

======== RESULTS of Greedy TakeandPutOne for n=50 using NNLS Optimization =======

Initial Order:  13  75  43  71  39  56  21  45  80  46  91  87  14  65  24  16  86  6  15  31  66  36  54  90  85
                   37  94  84  95  88  4  58  52  42  48  26  47  81  33  41  44  99  29  11  69  100  68  50  77  53
SSE: 1101940

True Order:  4  6  11  13  14  15  16  21  24  26  29  31  33  36  37  39  41  42  43  44  45  46  47  48  50
                  52  53  54  56  58  65  66  68  69  71  75  77  80  81  84  85  86  87  88  90  91  94  95  99  100
SSE: 75029.18

Estimated Order:  4  6  11  13  15  14  16  21  24  26  29  31  33  37  36  39  43  42  44  46  41  47  45  48  50
                    53  52  54  56  58  65  66  68  69  71  75  77  81  80  84  85  87  86  88  90  91  95  94  99  100
SSE: 74753.38

Number of Iterations: 137
Number of Permutations Tried: 8363
True Sigma = 8
Estimated Sigma = 7.972816

Estimated Betas:

\[
\begin{array}{ccccccccccccc}
[1] & 4.3786 & 6.0286 & 0.2484 & 3.0540 & 0.5156 & 0.0000 & 5.6590 & 1.8010 & 2.4513 & 3.6068 & 0.0000 & 4.9416 & 0.7938 \\
[14] & 1.2232 & 2.8403 & 3.2715 & 0.1322 & 1.5151 & 0.5031 & 0.0615 & 1.1686 & 0.4442 & 1.5875 & 1.6755 & 0.8434 & 0.6293 \\
[27] & 3.5731 & 0.5614 & 3.2313 & 5.3477 & 2.1655 & 0.4251 & 3.0573 & 0.8872 & 4.3082 & 2.7436 & 0.2441 & 3.3290 \\
[40] & 1.5414 & 0.9938 & 0.8447 & 1.0030 & 1.9635 & 0.1328 & 4.2130 & 0.8849 & 4.6321 & 0.3599 \\
\end{array}
\]

Spearman Rank Correlation between True and Estimated Orderings: 0.9979832
Euclidean Distance between True and Estimated Orderings: 42

We observe that both methods obtained very good sequencing results, with the NNLS optimization using SSE performing slightly better than the Closest Peers method (Spearman correlations between the true and estimated orderings for NNLS method is 0.998 and for Closest Peers is 0.995). Both correlation values are extremely high in magnitude; for \( n = 50 \) elements, such values are very rare to occur by pure chance. The Closest Peers method has misplaced or switched some elements, such as 11 & 13; 21 & 24; 36, 37, & 39, 80 & 81, etc. since they are close to one another and \( \sigma \) was not very small. The NNLS method gives a better estimated ordering with elements placed nearly in their correct order aside from the switching of some consecutive elements such as 14 & 15 and 94 & 95, etc. In both cases, the estimated orderings have criterion values that are slightly smaller than those of the true orderings, resulting in a slight underestimation of \( \sigma \) for the NNLS method. The values of the estimated spacings are also close to the true adjacent spacings, especially considering that a value of \( \sigma = 8 \) was chosen when generating the pairwise distances. Our experimentation has shown that this example is not an isolated one and that for the most part, NNLS performs better than Closest Peers although they both do quite well in sequencing with typical Spearman correlations between the estimated and true orderings of at least 0.80 for Closest Peers and at least 0.90 for NNLS even for high values of \( \sigma \) (such as \( \sigma = 20 \)).

As anticipated in Section 3.10, the SSE objective function performs better than ADP in this case, since the distances were generated from Model 3.3. The Closest Peers Method for sequencing could
result in an improvement over SSE when applied to real document collections. Such examples are given in Chapter 6.

In this section, the use of the Take and Put One algorithm was demonstrated using the SSE and ADP criterion functions and it is evident that the algorithm works reasonably well in both cases. Possible ways to further improve the Take and Put One algorithm are discussed in the following section.

### 4.4 Improvements to the “Greedy Take and Put One” Algorithm

As is, the “Take and Put One” method for permuting elements works reasonably well. However, there some variations and possible improvements that could be made. For instance, if prior information, such as an estimate of the ordering from another method, or some external indication about the ordering, is available, then we could incorporate this into the initial permutation in the algorithm rather than starting at a purely random one. This will increase the performance as well as the efficiency of the algorithm.

Another possible way to try to improve the algorithm is via Simulated Annealing (Kirkpatrick et al, 1983), which is an adaptation of the Metropolis-Hastings MCMC algorithm (Metropolis et al, 1953). The simulated annealing method starts at a random permutation and attempts to minimize a criterion function $\rho(\tau)$ (where we take $\rho(\tau) = SSE_{\text{con}}(\tau)$ or $ADP(\tau)$ in our case). But rather than rejecting permutations that lead to higher criterion function values, at each iteration the algorithm decides probabilistically whether to move to the new permutation, or to stay at the current one, choosing probabilities that hopefully will lead to lower criterion function values in the long run. The acceptance probability function, $P(e_{\text{current}}, e_{\text{next}}, T)$, depends on the energies $e_{\text{current}} \equiv \rho(\text{current})$ and $e_{\text{next}} \equiv \rho(\text{next})$ of the two permutations (states) and on a global parameter $T$ called the temperature. $P$ must be positive even when $e_{\text{next}} > e_{\text{current}}$ in order to avoid the algorithm from remaining stuck at a local minimum. As $T$ approaches 0, $P(e_{\text{current}}, e_{\text{next}}, T)$ should also approach 0 when $e_{\text{next}} > e_{\text{current}}$, and should remain positive otherwise. Sufficiently small values of $T$ tend
to increasingly move to states with lower energies. The case $T = 0$ corresponds to the “Greedy Take and Put One” algorithm described above. Note that in simulated annealing, we need not have $P(e_{\text{current}}, e_{\text{next}}, T) = 1$ when $e_{\text{next}} < e_{\text{current}}$, that is, we do not always have to move to a lower energy state when such is found. Usually, $P$ is chosen so that the probability of accepting a move decreases when the difference $e_{\text{next}} - e_{\text{current}}$ increases (i.e., small uphill moves are allowed to be more likely than large ones). These last two conditions are not required for the method to work although most implementations of the algorithm do use them. The other mentioned conditions on $P$ are necessary. Simulated annealing is advantageous as it can be adapted to any optimization problem and unlike many other methods has the ability (in principle) to reach global minima. It is a robust technique that can generally enhance performance. However, like most optimization algorithms, it is sensitive to the choice of parameters and requires proper fine-tuning.

In the next section, we introduce another variation of the Take and Put One algorithm that is useful for data that consists of a training set.

### 4.5 Greedy “Fix T Take and Put One” Algorithm

In certain contexts, the order of some of the elements may already be known, for example, when the documents are separated into a training set having known dates or known order, and a test set. We can use what we call the “Fix T Take and Put One” algorithm to estimate the ordering for the documents in the entire corpus by fixing the order of the documents in the training set. This will allow for more efficient computation than the regular “Take and Put One” since the algorithm avoids checking some of the permutations that we know are not correct.

The “Greedy Fix T Take and Put One” algorithm first separates the elements into two sets: one set has $T$ elements with the order fixed and the other set has $n - T$ elements with unknown order. The algorithm moves among permutations by randomly removing one of the $n - T$ unfixed elements at a time, and randomly placing it into a new position in the permutation, while keeping the relative order of the other elements fixed. In this particular method, we do not ‘take’ any of the $T$ points
with fixed order, we only move the elements in the test set. It is “greedy” in the sense that it makes the locally most optimal choice for the criterion value at each stage without carrying out the computations for all possible orderings. Thus, if the criterion is reduced by moving a particular element to another place, then that element is moved and kept there, whereas if moving that element to that new position increases the criterion value, the element is not moved. The algorithm works as follows (and is described below using SSE, although it can be applied to any criterion function):

“Greedy Fix T Take and Put One” Algorithm:

1. Start with an initial permutation where the $T$ elements from the training set are in their correct order and the remaining points are randomly placed anywhere else between the elements in the training set. Compute the SSE [denoted as $SSE_{\text{con}}(\text{current})$].

2. Create a $(n-T)(n-1) \times 2$ ‘pairs matrix’ where each row consists of element pairs $(i,j)$, where $i$ represents the position (in the current permutation) of an element from the test set that we will ‘take’ and $j$ represents the new position that we will ‘put’ the selected element into.

3. Randomly permute the rows of the pairs matrix. Select the first row of the pairs matrix, denoted as $(i,j)$.

4. Remove the element that is in the $i$th position from the current permutation and place it into the $j$th position (while keeping the relative order of all the other points fixed). Compute the new SSE [denoted as $SSE_{\text{con}}(\text{next})$].

5. If $SSE_{\text{con}}(\text{next}) < SSE_{\text{con}}(\text{current})$, then keep the element in the new position and let current=next. Go back to Step 3 and continue. Else, leave the current permutation as is and select the next row in the pairs matrix, denoted as $(i,j)$. Go back to Step 4 and continue.

6. Continue the above steps until you have reached the bottom of the pairs matrix but still have not reduced the SSE of the current permutation. (i.e. stop when each of the $(n-T)(n-1)$
different possibilities obtained by removing an element from the current permutation and placing it somewhere else has a higher SSE than the current permutation).

One iteration of the above algorithm is completed when a new permutation is accepted (i.e. the next permutation results in a lower SSE than the current permutation). Hence, in one iteration this algorithm will try at most \((n - T)(n - 1)\) possible permutations and of course it will try all \((n - T)(n - 1)\) take and put one possibilities on the current permutation on the last iteration before stopping. In this algorithm, we must recompute the pairs matrix each time a new permutation is accepted (whereas the indices in the pairs matrix stays fixed for the “Take and Put One” algorithm). This is because we need to keep track of the positions of the elements in the test set which will change after each iteration since we are only allowed to move elements from the test set. Here again, the algorithm may not converge to the global minimum. Although this algorithm is more computationally intensive than the “Take and Put One” method since it recreates the pairs matrix on each iteration, it will ultimately try fewer number of permutations per iteration (since the order of the elements in the training set is fixed), making it more efficient in this manner.

At the end of the “Fix T Take and Put One” procedure, we will obtain the estimated optimal ordering that minimizes the criterion function and has \(SSE = SSE_{\text{con}}(\text{opt})\). As before, the error variance is estimated by \(\hat{\sigma}^2 = \left(\frac{n - 1}{2}\right)^{-1} SSE_{\text{con}}(\text{opt})\). This algorithm results in an overall estimated ordering, ‘opt’, of all the elements. The ordering of the elements in the test set can be obtained by simply removing the elements in the training set from the overall estimated ordering.

Examples which pertain to the use of this algorithm will be applied to the DEEDS data sets with a training and test set of documents in Chapter 6.

This chapter has discussed general optimization algorithms for sequencing elements and focused on one method (with possible variations) to estimate an optimal ordering. The next chapter describes an optimization approach based on MCMC as an alternative to the “Greedy Take and Put One” algorithm or other such algorithms.
Chapter 5

Estimation and Inference on Sequencing via MCMC

An alternative to the algorithms described in Chapter 4 can be based on MCMC methods. One advantage of the MCMC approach is that it is generally robust provided reasonable model assumptions are made. Another advantage is that it allows for natural confidence statements regarding the ordering of selected subsets of the elements (to be discussed in Sections 5.4 and 5.5).

5.1 Markov Chain Monte Carlo Methods

Markov Chain Monte Carlo (MCMC) methods are most commonly used in Bayesian statistics to sample from a target distribution (i.e. a probability distribution of interest, typically a posterior distribution) that is difficult to sample from directly. The Ergodic Theorem tell us that under certain conditions, a Markov chain will converge to its stationary distribution. The idea behind MCMC is that if one could create a Markov chain whose unique stationary distribution is the desired target distribution, then after a sufficiently large number of steps, this chain will approach that target distribution. We could then run the chain long enough (discarding initial values so as to allow the chain to converge to its limiting distribution) and the resulting states can then be viewed
as an approximate sample from the target distribution. These states will of course not form an iid sample, but averages from this sample can nevertheless be used for purposes of estimation. The actual Markov chain can be constructed using various algorithms, one of which will be proposed in Section 5.3.

In order to perform inference procedures for sequencing problems via the MCMC, it is useful to introduce a timeline model for the elements rather than working solely with their permutations. This timeline, denoted by \( \tilde{a} \), is a vector of positions of the elements relative to the designated first element. Such a timeline vector provides more information than a permutation, as it gives not only the order of the elements but also indicates how far apart they are from one another. We will implement an MCMC method using a specific posterior distribution for \( \tilde{a} \) and for a scaling parameter \( \sigma \) (to be defined in Section 5.2) as our target distribution. The goal is then to sample from this posterior distribution; these samples will allow us to produce estimates for the ordering of the elements and for \( \sigma \), as well as to perform various other inference procedures. First, however we introduce a Bayesian framework for this problem, and so establish an appropriate likelihood function and priors in order to calculate a posterior distribution.

5.2 Likelihood, Priors, and Posterior Distributions

Before proceeding, we introduce some required notation.

Notation:

- \( n \) = the number of documents (i.e. elements).
- \( \tilde{a} = (a_2, \ldots, a_n) \) = timeline vector. Here, \( a_i = t_i - t_1 \), for \( i = 1, 2, \ldots, n \) where \( t_i \) is the time position of element \( i \), and \( a_i \) is the position of item \( i \) relative to item 1 (the reference point) with \( a_1 = 0 \). Note: \( a_i \) can be negative if the selected reference point is not the first point in the true ordering of the elements.
- \( r \equiv \binom{n}{2} \) = the number of pairwise distances between documents (elements).
Using this timeline vector, we propose the following distribution for the pairwise distances, $Y_{ij}$:

$$
\mathcal{L} \left( Y_{ij} \mid a, \sigma^2 \right) \sim |a_j - a_i| + e_{ij} ; \text{ where } e_{ij} \text{ are iid } N(0, \sigma^2).
$$

The likelihood function then becomes:

$$
L(a, \sigma \mid Y_{ij}) = \frac{1}{(2\pi)^{r/2} \sigma^r} \exp \left\{ -\frac{1}{2\sigma^2} \sum_{i<j} (Y_{ij} - |a_j - a_i|)^2 \right\} ; \quad (5.2.1)
$$

where $a_1 = 0$, and each $a_i \in \mathbb{R}$, for $i = 2, \ldots, n$, and $\sigma > 0$. Note that this likelihood function is related to the regression model (3.3.1) that was presented in Section 3.3. From the regression model (3.3.1), and provided that $(1, 2, \ldots, n)$ represents the correct order of the elements, $Y_{ij} \sim \text{independent } N \left( \sum_{k=i}^{j-1} \beta_{k,k+1}, \sigma^2 \right)$, where the $\beta$s are the spacings between consecutive elements in the ordering. When the order is correct we will have: $a_j - a_i = \sum_{k=i}^{j-1} \beta_{k,k+1} > 0$ for $j > i$ so that our likelihood function expressed in terms of the timeline is just a reparametrization of the likelihood expressed in terms of the spacings $\beta_{k,k+1}$. However, an important difference is that the likelihood (5.2.1) does not assume that the elements are placed in their correct order, thus allowing the $a_i$ to possibly be in non-increasing order, or even to be negative, as noted above.

We propose to use a flat prior on the $\{a_i\}$, and an Inverse-Gamma($\alpha, \beta$) prior distribution on $\sigma^2$, this latter being the conjugate prior (for $\sigma^2$) when using the Normal likelihood.

**Recall:** The Inverse Gamma($\alpha, \beta$) density is:

$$
f(w; \alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} w^{-\alpha - 1} \exp\{-\beta/w\} ; \; w > 0
$$

where $\alpha > 0$ is a shape parameter and $\beta > 0$ is a scale parameter. This density has mean and variance:

$$
E(W) = \frac{\beta}{\alpha - 1}, \quad \text{for } \alpha > 1, \quad \text{and} \quad \text{Var}(W) = \frac{\beta^2}{(\alpha - 1)^2(\alpha - 2)}, \quad \text{for } \alpha > 2.
$$
Note: The hyperparameters $\alpha$ and $\beta$ can be chosen so that the Inverse Gamma prior is relatively flat (so as to not give too much importance to any specific values of $\sigma^2$). This suggests using the values of $\alpha = -1$ and $\beta = 0$ which would then make it an improper prior. Alternatively, if a proper prior is preferred, one could choose the hyperparameters $\alpha$ and $\beta$ so that the mean of the Inverse Gamma prior is equal to some reasonable estimate of the variance, $\hat{\sigma}^2$, relative to the time interval over which we are sequencing, or is equal to some other reasonable estimate of the variance. In order to make the prior relatively flat, we need to fix the mean to $\hat{\sigma}^2$, and make the variance large, resulting in choices $\alpha \downarrow 2$, and $\beta = \hat{\sigma}^2(\alpha - 1)$. We choose relatively flat priors as we typically will not have any information about the time gap between documents or the value of $\sigma$. If such information was available, informative priors can be selected instead.

Let $\pi$ denote the resulting Posterior Distribution under this model. The posterior distribution of $\sigma^2$ and $g$ is proportional to the product of the likelihood (5.2.1) and our Inverse Gamma prior. The posterior, $\pi$, does not have a simple recognizable distribution. However, the conditional posterior distribution of $\sigma^2$, given $\alpha$, $\beta$, and $g$, is Inv-Gamma \( \left( \alpha + \frac{r}{2}, \beta + \frac{1}{2} \sum_{i<j} (Y_{ij} - |a_j - a_i|)^2 \right) \). This computation is easily verified since we have used the conjugate prior for mathematical convenience.

Now that the posterior distribution has been determined, we can proceed to introduce a particular MCMC algorithm to sample from that distribution.

5.3 The Metropolis-within-Gibbs Algorithm

The “Metropolis-within-Gibbs Algorithm” is a specific MCMC algorithm used to sample from multidimensional probability distributions. This is convenient in our case as we are trying to sample from the posterior distribution of $n$ parameters/coordinates namely the $(a_2, \ldots, a_n, \text{and } \sigma^2)$. The general idea is to start at some initial value for all coordinates and, one at a time, propose a new value for each coordinate using some symmetrical proposal/jumping distribution (while keeping all other coordinates fixed). The “proposal/jumping distribution”, $q(\theta_1, \theta_2)$ say, specifies probabilities of moving from one proposed value $\theta_1$ of a coordinate to another value
θ_2, and “symmetrical” means that q(θ_1, θ_2) = q(θ_2, θ_1). The “acceptance ratio” A is defined as the ratio of the target distribution evaluated at the new value to the target distribution evaluated at the previous value. This acceptance ratio A is calculated, and if A ≥ 1, we accept the new proposed value. Otherwise, we accept the new proposed value with probability A. The Metropolis-within-Gibbs algorithm actually creates a reversible Markov chain with respect to the target distribution, π, meaning that π_i p_{ij} = π_j p_{ji} for all states i and j, where p_{ij} represents the transition probability from state i to j. Essentially, reversibility means that the Markov chain and its reverse (the chain run backwards in time) will have the same distribution. This condition ensures that the target π is the stationary distribution, and hence that the chain converges to the target distribution (Hastings, 1970).

We apply the above Metropolis-within-Gibbs algorithm in order to sample from the posterior distribution for σ^2 and a.

**Initial Values:**

There are evidently different approaches to choosing hyperparameters and initial values. One way is to use random starting values for all coordinates. Another approach is to use the results from the “Greedy Take and Put One” algorithm with SSE criterion or to use some other reasonable estimates if some prior information is available. Strictly speaking, the choice of the initial values should not matter as the Markov chain should converge to the same distribution regardless of the starting position. However, one may prefer the second approach to choosing initial values as one would expect it to lead to faster convergence. Initial values for both of these approaches are outlined below:

**Initial Values: Using Random starting values**

- Randomly generate \( \hat{\sigma} \) from a Uniform(0, R/4) or Uniform(0, R), where R is the range for the time period over which we are sequencing. (The time between elements is approximated here to have a standard deviation of approximately R/4, but since σ is the standard deviation of the pairwise
distances, and would be expected to be higher, we can allow for a larger range. Other reasonable estimates, or a range for the value of \( \sigma \), could be used.)

- Choose the first element or a random one to be item 1 (the reference point).
- Relabel the elements according to the reference point, and let \( \hat{a}_1 = 0 \). Randomly generate each \( \hat{a}_i \) from Uniform\((-R,R)\) where \( R \) is the range for the time period we are sequencing over. (This seems reasonable as the time difference between any two elements must be between \(-R\) and \(R\)).
- As indicated in Section 5.2, the hyperparameters are chosen to have values \( \alpha \downarrow 2 \) and \( \beta = \hat{\sigma}^2(\alpha - 1) \).

**Initial Values:** Using information from the “Greedy Take and Put One” Algorithm with SSE Criterion

Recall that the Closest Peers method described in Section 3.10 is not based on a parametric model and so results from the Take and Put One method using the ADP criterion cannot give initial values for \( \sigma \) or any of the \( a_i \). However, the NNLS method is based on an appropriate regression model on the pairwise distances between elements (refer to Section 3.3 and 3.5) and therefore the results from the Take and Put One algorithm using the SSE criterion can be used to give starting values as follows:

- Use \( \hat{\sigma}^2 = \left( \frac{n-1}{2} \right)^{-1} SSE_{con}(opt) \) from the “Take and Put One” algorithm as the initial value for \( \sigma^2 \).
- Choose the first item in the estimated ordering from the sequencing algorithm to be item 1 (the reference point), i.e. choose the first item according to the direction of points given by the estimated ordering from the sequencing algorithm.
- Relabel the elements according to the order of the estimated optimal ordering. Use the estimated \( \beta \)s corresponding to this estimated ordering to obtain initial values for \( a_i \), namely: \( \hat{a}_i = \sum_{k=1}^{i-1} \hat{\beta}_{k,k+1} \) for \( i = 2, \ldots, n \) and \( \hat{a}_1 = 0 \).
- As indicated in Section 5.2, the hyperparameters are again chosen to have values \( \alpha \downarrow 2 \) and \( \beta = \hat{\sigma}^2(\alpha - 1) \).

The algorithm is described below step by step, where the posterior distribution \( \pi \) is the target distribution we wish to sample from.
**Metropolis-within-Gibbs Algorithm:**

1. Given an initial value for \((\sigma^2, a_2, \ldots, a_n)\), choose \(\sigma^2\), and then one coordinate \(a_i\) at a time, for convenience. The actual order in which coordinates are selected should not matter. (We have chosen the coordinates systematically on each iteration in our implementation.)

2. Propose a new \(\sigma^2_{\text{new}} \sim N(\sigma^2, \tau^2)\) (where \(\tau^2\) is to be defined below) and let \(U \sim \text{Uniform}(0, 1)\):
   - If \(\sigma^2_{\text{new}} < 0\), set \(\pi(a_2, \ldots, a_n, \sigma^2_{\text{new}}) = 0\) (as we cannot have negative values for \(\sigma^2\), and this condition will reject whenever \(\sigma^2_{\text{new}} < 0\)).
   - If \(U < \frac{\pi(a_2, \ldots, a_n, \sigma^2_{\text{new}})}{\pi(a_2, \ldots, a_n, \sigma^2)}\) then accept \(\sigma^2_{\text{new}}\). Else, reject \(\sigma^2_{\text{new}}\) and leave \(\sigma^2\) as is. All other coordinates remain fixed.

3. Propose, one at a time, a new \(a_i' \sim N(a_i, \eta^2)\) (where \(\eta^2\) is to be defined below) and let \(U \sim \text{Uniform}(0, 1)\):
   - If \(U < \frac{\pi(a_2, \ldots, a_i', \ldots, a_n, \sigma^2)}{\pi(a_2, \ldots, a_i, \ldots, a_n, \sigma^2)}\) then accept \(a_i'\). Else, leave \(a_i\) as is. All other coordinates remain fixed.

4. Repeat for each coordinate.

5. Record the updated set of coordinates \((\sigma^2, a_2, \ldots, a_n)\).

The standard deviations, \(\tau\) and \(\eta\) for the proposal/jumping distributions above are held constant and are meant to be chosen so that the chain mixes well (i.e. the rate at which we accept new proposals is not too large or small). If the values of \(\tau\) and \(\eta\) are too small, the algorithm will accept new proposals often but the samples will move around the parameter space slowly leading to slow convergence. Conversely, if the standard deviations are too large, the algorithm will reject more often because the chain will get stuck in regions of low posterior probability so that once again the chain will converge slowly. In practice, the values of \(\tau\) and \(\eta\) are fine-tuned for good results by experimentation.

Each iteration of the Metropolis-Within-Gibbs algorithm consists of steps 1-5 above, and results in updated values for \(\sigma^2\) and \(a\). We run the Metropolis-within-Gibbs algorithm for \(N\) iterations, for
some very large \( N \), and discard the initial \( B \) samples for “burn-in” to allow the chain to converge (where \( B \) is typically a few thousand or so). The distribution of these \( N - B \) generated samples should have converged to \( \pi \) and we can view the values from these \( N - B \) samples to be approximate random samples of \( \sigma^2 \) and \( \alpha \) from the posterior distribution.

Note that this procedure could be modified by directly using the permutation (rather than working with the timeline vector). In this case, the likelihood would be viewed as a function of the permutation and of \( \tilde{\beta} \) as in Model (3.3), and one could put a (flat) prior on the \( \beta \)s. As discussed before, we find it advantageous to work with the timeline vector as it provides more information and can help quantify how far apart the elements actually are. In addition, the generated timeline vectors could be used to check the accuracy of the estimated ordering obtained from the “Take and Put One” algorithm, or obtained from another method, or to perform other inferences to help quantify the uncertainty. This will be discussed in Section 5.5.

In the next section, we explain how to obtain an estimate of the true ordering and of \( \sigma \) using the generated samples from an MCMC algorithm, such as the Metropolis-within-Gibbs.

### 5.4 Sequencing and Interval Estimation via MCMC

A question of some interest is how one can make confidence statements about the permutation, or confidence intervals for particular timeline points. Another question of interest is how to estimate \( \sigma^2 \). This is of interest because an estimate of \( \sigma \) can provide us with some indication of how accurate our measures of distances are, and how accurate the estimated ordering of the elements is. The underlying difficulty here is that the true ordering is unknown so that we cannot use traditional methods to estimate \( \sigma \) since we may not have \( SSE \sim \sigma^2 \chi^2_\nu \), where \( \nu = \left(\frac{n - 1}{2}\right) \) as was noted in Section 3.6. We therefore use a Bayesian approach here for estimation.

We can use the samples from the posterior distribution, \( \pi \), to estimate the true ordering by means of the following procedure: Each generated sample of the timeline vector is first converted to
its corresponding permutation of the elements (i.e. the permutation that puts the values of $a_i$ in increasing order). Each of these different permutations (unique up to reversal) is recorded along with its estimated posterior probability, which is equal to the proportion of times that that particular permutation occurs in the posterior samples. The most frequently occurring permutation will be considered as the estimate for the true ordering, estimated along with its posterior probability. In addition, one could generate a confidence region for the true ordering by collecting several permutations, each of which have high posterior probability. Thus for example, a $100(1 - \alpha)\%$ confidence region would contain all of the most probable permutations that occur in at least $(1 - \alpha)$ proportion of the posterior samples. However, such confidence regions may not be sufficiently meaningful in practice. What would be of more interest is to find an interval estimate for a particular timeline point, $a_i$, which would allow us to quantify how far apart in time the element $i$ is to the reference point, and whether its position belongs before or after the reference point. Similarly, an interval estimate for the difference between two timeline points $a_j - a_i$ can be used to make inferences about the time between the $i$th and $j$th element, and interval estimates for ratios of timeline points $\frac{a_j}{a_i}$ and $\frac{a_k}{a_i}$ would be useful in comparing points relative to the reference point.

Using the samples from the posterior distribution, we can obtain credible intervals (i.e. intervals of high posterior density) for any of the $a_i$, or differences $a_j - a_i$, and/or for $\sigma^2$. A $100(1 - \alpha)\%$ credible interval for a parameter is an interval that contains $100(1 - \alpha)\%$ of the probability in the posterior distribution for that parameter. Credible intervals are not unique and so there are several possible approaches for obtaining such intervals. Some common approaches for constructing credible intervals for a scalar parameter are: (1) choosing the narrowest interval in the domain of the posterior distribution that contains the specified amount of probability; (2) choosing an interval so that equal probability lies above it and below it; and (3) choosing the interval for which the mean is located in the center (if the mean exists).

The main advantage to using Bayesian intervals (aside from the fact that there is no known Frequentist method to estimate $\sigma$) is that it provides for useful interpretation. In the Bayesian context, a $100(1 - \alpha)\%$ credible interval for a particular parameter means that the parameter lies within the calculated interval with $100(1 - \alpha)\%$ posterior probability. The identical statement cannot be
made with Frequentist confidence intervals.

5.5 Other Inferences

Aside from interval estimation, one can use the timelines vector to make specific inferences about the true ordering of the elements or of subsets of the elements. For example, one could estimate probabilities of events such as: (i) a candidate ordering is the true ordering; (ii) two specific elements are at least \( t \) units apart; (iii) two specific elements belong on the same side or on different sides of the reference point (keeping in mind that order reversal cannot be distinguished); (iv) some elements are placed out of order in an estimated ordering; or (v) a particular sub-ordering holds amongst several of the elements. Posterior probabilities for any such event can be estimated by the proportion of times that particular event occurs in the samples resulting from the MCMC algorithm. This is an important advantage of the MCMC approach.

Suppose, for example, that we want to estimate the accuracy of a candidate ordering, say one that was estimated by the “Take and Put One” sequencing algorithm or by some other method. Since we are using the first item in the estimated ordering as the reference point, we must have all \( a_i > 0 \) and in increasing order if the ordering is correct. If any of the \( a_i \) are negative, it suggests for that particular sample that the estimated ordering has placed the first item in an incorrect position. Similarly, if the \( a_i \) are decreasing, this would suggest that the sequencing algorithm has placed some elements out of order and we can then quantify how far away those elements have been misplaced by examining the values of their \( a_i \). For example, if all the timeline coordinates are in increasing order up to a certain point \( k \), and then \( a_k \) is smaller than the previous coordinate, this suggests that the \( k \)th element in the estimated ordering should come before where it had been placed by the sequencing method.

For each of the \( N - B \) samples, we can examine their timeline vector for such properties in order to estimate the posterior probability of any of the events mentioned above. Thus, the estimated
posterior probability of any event $A$ is just:

$$\hat{P}(A) = \frac{1}{N-B} \sum_{\ell=B+1}^{N} I(a_{\ell} \in A).$$

Given a triplet $(D_i, D_j, D_k)$ consisting of 3 documents or elements within the corpus, it is of particular interest to estimate the true ordering of these elements. Since order reversal cannot be distinguished, this is equivalent to estimating which document/element ($i$, $j$, or $k$) belongs in the middle position. To do this, we count the proportion of times in the posterior samples that $a_i$ is in the middle amongst the three timeline components, that $a_j$ is in the middle, and that $a_k$ is in the middle. The element corresponding to the largest of these proportions is estimated to belong in the middle with its respective posterior probability. We would expect the MCMC method to give posterior probabilities substantially greater than $1/3$ for the results to be considered meaningful.

The example below will illustrate the sequencing process via the MCMC approach as well as some of the inferences discussed above for $\sigma$ and for the ordering using simulated data.

### 5.6 Example: Sequencing using the MCMC Approach with Simulated Data

$n = 10$ points (denoted ‘numbers’) were randomly generated uniformly from the numbers 1-100 without replacement. The observed distances were calculated as the true Euclidean distances on $\mathbb{R}^1$ between each pair of elements + independent random $\mathcal{N}(0, 25)$ noise. First, we estimate the ordering of the points using the Greedy Take and Put One algorithm with SSE criterion (not using MCMC):

Numbers: 2 9 25 26 27 30 31 52 64 66

==== RESULTS of Greedy TakeandPutOne for n= 10 using NNLS Optimization =====
Initial Order: 66 25 26 64 27 31 52 30 9 2  
SSE: 10378.23

True Order: 2 9 25 26 27 30 31 52 64 66  
SSE: 789.6869

Estimated Order: 66 64 52 30 31 27 26 25 9 2  
SSE: 707.7323

Number of Iterations: 13  
Number of Permutations Tried: 198

True Sigma= 5  
Estimated Sigma= 4.433873

Estimated Betas:  
4.6114 11.6239 15.9512 4.0710 2.3117 1.8832 5.5662 11.9023 4.6606  
True Spacings:  
7 16 1 1 3 1 21 12 2

Spearman Rank Correlation calculated by R: -0.9878788  
Euclidean Distance between True and Estimated Orderings: 2

We can see that the Take and Put One algorithm here has interchanged elements 30 and 31 but otherwise that the sequencing is correct. The Spearman Rank correlation between the estimated and true orderings is quite high (-0.99) with the negative value showing that the estimated ordering was in the backwards direction. The squared Euclidean distance between the estimated and true orderings has a very low value of 2, indicating that the two orderings are very close to one another.
Also, the estimated $\beta$s are in the backwards direction of the true adjacent spacings because the estimated ordering was in reverse chronological order. Next, we apply the MCMC algorithm to check the accuracy of the ordering that has been estimated by the Take and Put One method. For instance, we know that the 5th element in the estimated ordering (i.e. the number 31) was placed in the wrong position and we can estimate this probability by counting the proportion of samples that have $a_5 < a_4$. Similarly, the probability that the estimated ordering is correct can be approximated by the proportion of samples that have all $a_i > 0$ and in increasing order. The probability that the first element in the estimated ordering belongs elsewhere is estimated by the proportion of timeline vectors that have any negative values. Results are given below based on $N = 10,000$ MCMC iterations with a burn-in of $B = 3000$:

- $\hat{P}(5\text{th element is misplaced}) = 0.14$
- $\hat{P}(4\text{th element is misplaced}) = 0$
- $\hat{P}(1\text{st element is misplaced}) = 0.06$
- $\hat{P}(\text{estimated ordering is correct}) = 0.395$.

The MCMC procedure indicated that it is quite certain (with approximately 94% probability) that the first element has not been placed out of order. The 5th element is estimated to have been placed out of order with about 14% probability, whereas the 4th element is estimated to be incorrectly placed with 0% probability. This occurs because the 4th element follows the order of the previous elements and the 5th one is the element that causes the order to broken. The probability that the full ordering estimated by the Take and Put One is correct is only approximately 40%. The MCMC procedure was able to detect that the full ordering is incorrect with 60% posterior probability, although it is only a minor interchange (of 30 and 31) that had actually caused the error.

It is worth noting that the MCMC procedure, in and of in itself, can be used to estimate the entire ordering (not just to check the accuracy of a given estimated ordering). Below we illustrate the use of the Metropolis-within-Gibbs algorithm with $N = 10,000$ iterations and a burn-in period of $B = 3,000$ to estimate the ordering of the elements from the example above and perform other
inferences using the following two different approaches for choosing the initial values:

1. Approach 1:
   Select a random initial value for $\theta$ and $\sigma$ (denoted $\hat{\sigma}$) and use hyperparameter values $\alpha = 2.001$ and $\beta = 1.001\hat{\sigma}^2$.

2. Approach 2:
   Use initial values corresponding to the estimated ordering (denoted ‘opt’) from the Take and Put One algorithm with SSE criterion: $\hat{\sigma}^2 = \left(\frac{n-1}{2}\right)^{-1} SSE_{con}(opt)$ and use the estimated $\beta$s to calculate the $\hat{a}_i = \sum_{k=1}^{i-1} \hat{\beta}_{k,k+1}$. Once again, we use hyperparameter values $\alpha = 2.001$ and $\beta = 1.001\hat{\sigma}^2$.

Then, we compute 95% and 99% credible intervals for $\sigma$ based on the MCMC samples as outlined in Section 5.4. Displayed below is the output:

===== RESULTS of METROPOLIS-WITHIN-GIBBS ALGORITHM Using Approach 1 for Initial Values=====

Number of iterations: N= 10000 Burn-in: B= 3000

Using a random timeline/sigma as initial value:
Initial value for timeline: 30.36862 -39.55631 7.330009 -54.92252 30.03173 0.3728934 34.62213 58.29134 9.999684
Initial value for sigma: 9.70419

Results based on 7,000 samples:

Estimated Ordering:
66 64 52 30 31 27 26 25 9 2 with estimated posterior probability = 0.3705714
Spearman Rank Correlation between True and Estimated Orderings: -0.9878788
Euclidean Distance between True and Estimated Orderings: 2

95% Credible Interval for sigma: ( 4.336239, 6.787102 )
99% Credible Interval for sigma: ( 4.245343, 7.503768 )

====== RESULTS of METROPOLIS-WITHIN-GIBBS ALGORITHM Using Approach 2 for Initial Values ======

Number of iterations: N= 10000 Burn-in: B= 3000

Using timeline/sigma values from Sequencing Algorithm as initial value:
Initial value for timeline: 4.611437 16.23531 32.18655 36.25756 38.56924
40.45239 46.01856 57.92086 62.5815
Initial value for sigma: 4.433873

Results based on 7,000 samples:

Estimated Ordering:
66 64 52 30 31 27 26 25 9 2 with estimated posterior probability = 0.358

Spearman Rank Correlation between True and Estimated Orderings: -0.9878788
Euclidean Distance between True and Estimated Orderings: 2

95% Credible Interval for sigma: ( 4.486152, 6.883445 )
99% Credible Interval for sigma: ( 4.233778, 7.399092 )

The computation time for both MCMC runs and the Take and Put One algorithm combined was less than 5 minutes in total on a Lenovo laptop with an i7-2640M CPU at 2.80 GHz. We observe in this case that both starting values resulted in the same estimated orderings (although
with slightly different posterior probabilities, which is due to Monte Carlo sampling error). In the many examples we have tested, we have found similar results. Specifically, the MCMC algorithm converges to the same estimated ordering when starting at different random initial values and values corresponding to the sequencing algorithm, with slightly different posterior probabilities. These results are consistent even with high values of \( \sigma \), but tend to have lower posterior probabilities. The posterior probabilities of around 0.36 to 0.37 in the above example represent the probability that the estimated ordering from the MCMC algorithm is the true ordering, which is much higher than \( 2/10! = 5.511464 \times 10^{-07} \) (which is the probability we would expect for a randomly estimated ordering). In this case, the NNLS sequencing algorithm as well as the Metropolis-Within-Gibbs algorithm runs resulted in the same estimated orderings. However, it may sometimes happen that both methods result in different estimated orderings. All credible intervals for \( \sigma \) (with 90% and 95% posterior probability) captured the true value of \( \sigma = 5 \).

In the next chapter, we will apply MCMC-based inferences as well as the other techniques we have discussed in this thesis in an attempt to sequence real sets of documents.
Chapter 6

Some Applications

In this chapter, we illustrate the application of the methods of Chapters 2 through 5 using real collections of documents and various distance measures between document pairs. The corpora that we experiment with are diverse with respect to the length of the documents, the topics, the time range over which the documents were written, and the language used (English, Latin, and translations from other languages such as Greek). Some of these collections contain documents on the same or similar topics, and some collections contain drafts of the same work. As noted in Section 2.1, documents and drafts of the same manuscript need to be treated differently in the types of distance measures used for them. Levenshtein distance is particularly useful for drafts of the same work whereas LSA-based distances and tableaux distances are more appropriate for use between pairs of documents. Each corpus will be described in detail in its section.

6.1 Sequencing Drafts

In this section, we apply our sequencing techniques to drafts or versions of the same document: In particular, we examine three different collections of drafts. As described in Section 2.1, we will use the normalized Levenshtein distance to measure similarity between pairs of drafts.
6.1.1 Netflix Paper Drafts

The first corpus contains 13 drafts of the introductory section to a paper titled “Statistical Significance of the Netflix Challenge” by Feuerverger et al (2012). The lengths of the different versions range between 890 to 1714 words. Naturally, later drafts tend to have longer lengths, however the lengths are not strictly monotone with respect to their temporal order. The normalized Levenshtein distance was calculated between each pair of drafts and is displayed in a matrix in the output below. The drafts are labelled according to their known true order and are placed in this true order in the distance matrix for the reader’s convenience. Although the order is known, the time range over which the documents were written is unknown for this data set. Version 2 happens to be a duplicate of Version 1 in this case (as can be seen from the distance of 0 in the matrix below), but we have kept both drafts in the collection for purposes of illustration.

The output below shows the following:

(a) Lengths of the drafts, and the matrix of normalized Levenshtein distances between document pairs.

(b) Sequencing results using the Greedy Take and Put One algorithm with the SSE criterion function.

(c) Results of the Metropolis-within-Gibbs algorithm with $N = 1,000,000$ iterations and a burn-in period of $B = 200,000$, starting from the parameter values generated by the sequencing algorithm in (b). i.e. $\hat{\sigma}^2 = \left(\frac{n - 1}{2}\right)^{-1} SSE_{con}(\text{opt})$ and $\hat{a}_i = \sum_{k=1}^{i-1} \hat{\beta}_{k,k+1}$ for $i = 2, \ldots, n$, where the elements are relabelled according to the estimated ordering, ‘opt’, from (b) and the SSE and $\beta$s correspond to the values from that estimated ordering. Following the guidelines from Section 5.3, we selected $\alpha = 2.001$ and $\beta = 1.001\hat{\sigma}^2$ as the values for the hyperparameters.

(d) 95% and 99% credible intervals for $\sigma$ and for the $a_i$ timeline points determined from (c).

(e) Results for sequencing triplets of documents for a number of triplets. For each such given
set of 3 documents, the estimated ordering (up to reversal) is shown, along with its posterior probability. Also displayed is a matrix containing each possible ordering of the 3 given elements along with the proportion of samples (amongst the 800,000 samples left after burn-in) that agreed with that particular ordering.

---------------

SEQUENCING RESULTS FOR THE NETFLIX DRAFTS
---------------

(a)

Document Lengths (presented in true ordering):

890 890 1064 1159 1160 1580 1621 1561 1686 1712 1678 1714 1685

Levenshtein Distance Matrix (presented in true ordering):

[1,] 0.0000 0.0000 0.3629 0.7273 0.7276 0.7334 0.7371 0.7369 0.7439 0.7451 0.7430 0.7441 0.7437
[2,] 0.0000 0.0000 0.3629 0.7273 0.7276 0.7334 0.7371 0.7369 0.7439 0.7451 0.7430 0.7441 0.7437
[3,] 0.3629 0.3629 0.0000 0.7093 0.7096 0.7018 0.7059 0.7063 0.7092 0.7095 0.7119 0.7111
[4,] 0.7273 0.7273 0.7093 0.0000 0.0024 0.4543 0.4691 0.4909 0.5543 0.5703 0.5740 0.5896 0.5874
[5,] 0.7276 0.7276 0.7096 0.0024 0.0000 0.4540 0.4688 0.4906 0.5541 0.5701 0.5738 0.5898 0.5876
[6,] 0.7334 0.7334 0.7018 0.4543 0.4540 0.0000 0.0291 0.2287 0.3306 0.3667 0.3839 0.4146 0.4104
[7,] 0.7371 0.7371 0.7059 0.4691 0.4688 0.0291 0.0000 0.2181 0.3295 0.3671 0.3858 0.4161 0.4139
[8,] 0.7369 0.7369 0.7063 0.4909 0.4906 0.2287 0.2181 0.0000 0.2032 0.2550 0.2849 0.3196 0.3194
[9,] 0.7439 0.7439 0.7063 0.5543 0.5541 0.3306 0.3295 0.2032 0.0000 0.0739 0.1019 0.1751 0.1927
[10,] 0.7451 0.7451 0.7092 0.5703 0.5701 0.3667 0.3671 0.2550 0.0739 0.0000 0.0301 0.1122 0.1278
[11,] 0.7430 0.7430 0.7095 0.5740 0.5738 0.3839 0.3858 0.2849 0.1019 0.0301 0.0000 0.0831 0.1002
[12,] 0.7441 0.7441 0.7119 0.5896 0.5898 0.4146 0.4161 0.3196 0.1751 0.1122 0.0831 0.0000 0.0155
[13,] 0.7437 0.7437 0.7111 0.5874 0.5876 0.4104 0.4139 0.3194 0.1927 0.1278 0.1002 0.0155 0.0000

(b)

======== RESULTS of Greedy TakeandPutOne for n= 13 using NNLS Optimization ========
Initial Order: 1 6 13 9 10 11 3 5 8 7 4 2 12
SSE: 8.995288

True Order: 1 2 3 4 5 6 7 8 9 10 11 12 13
SSE: 1.857761

Estimated Order: 12 13 11 10 9 8 7 6 5 4 3 1 2
SSE: 1.857943

Estimated Sigma= 0.1677816

Estimated Betas:
[1] 0.0000 0.1069 0.3251 0.0005 0.2334 0.0080
[7] 0.0931 0.1074 0.0392 0.0165 0.0440 0.0000

Spearman Rank Correlation between True and Estimated Orderings: \(-0.9890\)
Euclidean Distance between True and Estimated Orderings: 4

Firstly, we observe here that the estimated ordering from the Take and Put One sequencing algorithm is nearly perfect (with a Spearman correlation of \(\approx -0.99\)), aside from the interchanging of the consecutive drafts 12 and 13. Recall that drafts 1 and 2 are identical, so the interchanging of those two does not matter. Also, note that the the Spearman correlation is in reality higher than reported in the output since versions 1 and 2 are the same but are treated as being different when calculating the Spearman correlation. In this particular case, the true ordering has a slightly lower SSE than the estimated one and so the algorithm did not arrive at the global minimum and so we know that \(\sigma\) will be overestimated. Clearly the sequencing algorithm is taking into account characteristics of the contents of the drafts to estimate the ordering and is not simply using the lengths of documents. If it were just using the lengths of the documents, it would have placed versions 8 and 11 earlier in the estimated ordering than where they actually belong due to the lack
of monotonicity in the document lengths. However, the algorithm did overall do quite a remarkable job of sequencing these drafts. The estimated adjacent spacing \( \hat{\beta}_{1,2} = 0 \) as expected since drafts 1 and 2 are identical. We also have \( \hat{\beta}_{12,13} = 0 \) which indicates that the algorithm could not distinguish between the 12th and 13th drafts and explains why they have been switched in the estimated ordering. Also, \( \hat{\beta}_{4,5} \approx 0.0005 \) is small compared to other spacings, suggesting that drafts 4 and 5 may have been written closer in time than other pairs of drafts in this set. The \( \beta \) values are small but cannot be interpreted exactly since the time units are unknown in this case, but can be used to compare durations between consecutive pairs of drafts.

Next we present the results of the Metropolis-within-Gibbs algorithm that used the \( \sigma \) and timeline values generated from the sequencing algorithm above as initial values. It was based on 1,000,000 iterations in all with a burn-in of 200,000:

(c)

============ RESULTS of METROPOLIS-WITHIN-GIBBS ALGORITHM ============

Number of iterations: \( N = 1e+06 \) Burn-in: \( B = 2e+05 \)

Using timeline/sigma values from Sequencing Algorithm as initial value

Initial value for timeline:

0 0.1069 0.4321 0.4326 0.6660 0.6740 0.7670 0.8744 0.9136 0.9301 0.9742 0.9742

Initial value for sigma: 0.1677816

RESULTS BASED ON 800,000 SAMPLES:

Estimated Ordering by MCMC:

13 12 11 10 9 8 7 6 4 5 3 1 2 with estimated posterior probability = 0.0304
Spearman Rank Correlation between True and Estimated Orderings: -0.9890
Euclidean Distance between True and Estimated Orderings: 4

(d)
95% Credible Interval for \( \sigma \): (0.3873035, 0.537584)
99% Credible Interval for \( \sigma \): (0.3663374, 0.5733206)

95% Credible Interval for \( a_2 \): (-0.4254366, 0.4072765)
95% Credible Interval for \( a_3 \): (-0.5207107, 0.530518)
95% Credible Interval for \( a_4 \): (-1.203014, 1.308653)
95% Credible Interval for \( a_5 \): (-1.212389, 1.303234)
95% Credible Interval for \( a_6 \): (-1.11879, 1.162519)
95% Credible Interval for \( a_7 \): (-1.099456, 1.189248)
95% Credible Interval for \( a_8 \): (-1.090806, 1.160761)
95% Credible Interval for \( a_9 \): (-1.079903, 1.161842)
95% Credible Interval for \( a_{10} \): (-1.074981, 1.166466)
95% Credible Interval for \( a_{11} \): (-1.083686, 1.172206)
95% Credible Interval for \( a_{12} \): (-1.093173, 1.192736)
95% Credible Interval for \( a_{13} \): (-1.114348, 1.161954)

In this particular example, the MCMC method estimated a different ordering than the sequencing algorithm. The MCMC interchanged 1 and 2 as it could not distinguish between them (since they are in fact identical), and also switched the order of 4 and 5, but otherwise the ordering is correct. The MCMC has improved on the sequencing by correctly ordering drafts 12 and 13 but has instead switched 4 and 5. The Spearman correlation should be more than the 0.98 that is displayed in the output and the Euclidean distance should actually be 2, not 4 (since drafts 1 and 2 are the same). So, the MCMC ordering is equally good as the estimated ordering from the Take and Put One algorithm. The estimated posterior probability for the MCMC ordering is
about 0.03, which is of course greater than the value of \(2/13!\) based on a purely random ordering. For a set \(C\) belonging to the parameter space, the **Relative Belief Ratio** (Baskurt and Evans, 2013), \(RB(C)\), is defined as the ratio of the posterior to the prior probability of \(C\). This relative belief ratio is a measure of how beliefs in the true value being \(C\) have changed from a priori to a posteriori. For this particular example, the relative belief ratio of the MCMC ordering is \(RB(\tau_{\text{MCMC}}) = 0.0304/(2/13!) = 94650716\. This shows a large increase in the belief that the MCMC ordering is the true one from prior to posterior.

The following are results from sequencing various sets of triplets based on the MCMC samples:

(e)

```r
> ProbabilityForTriples(c(1,2,3),countmatrix)
Triple Order: 3 1 2 with estimated posterior probability = 0.467
[1,] 3 1 2 0.470
[2,] 1 2 3 0.467
[3,] 1 3 2 0.063
```

```r
> ProbabilityForTriples(c(1,2,12),countmatrix)
Triple Order: 1 2 12 with estimated posterior probability = 0.505
[1,] 1 2 12 0.505
[2,] 12 1 2 0.495
[3,] 1 12 2 0.000
```

The MCMC method accurately sequences the triplets of this data set. For example, in all the triplets involving both drafts 1 and 2, the order of 1 and 2 is interchanged approximately 50% of the time as we would expect since they are in fact identical drafts. In the triplet (1,2,3), 3 has a very small probability of being placed in the middle whereas draft numbers that are far from 1 and 2 (such as 12) have 0 or nearly 0 probability of being placed in the middle of the triplet.
Below are results for some triplets in which the drafts are spaced apart:

> ProbabilityForTriples(c(1,5,10),countmatrix)

Triple Order: 1 5 10 with estimated posterior probability = 1

[1,]  1   5  10  1.000
[2,]  1  10   5  1.000
[3,] 10   1   5  1.000

> ProbabilityForTriples(c(1,5,13),countmatrix)

Triple Order: 1 5 13 with estimated posterior probability = 1

[1,]  1   5  13  1.000
[2,]  1  13   5  1.000
[3,] 13   1   5  1.000

> ProbabilityForTriples(c(4,10,13),countmatrix)

Triple Order: 4 10 13 with estimated posterior probability = 0.833

[1,]  4  10  13 0.833
[2,]  4  13  10 0.167
[3,] 13  10  10 0.000

> ProbabilityForTriples(c(7,10,12),countmatrix)

Triple Order: 12 10 7 with estimated posterior probability = 0.950

[1,] 12  10   7 0.950
[2,]  7  12  10 0.050
[3,] 12  10  10 0.000
> ProbabilityForTriples(c(3,13,10),countmatrix)

Triple Order: 3 10 13 with estimated posterior probability = 0.833

[1,] 3 10 13 0.833
[2,] 3 13 10 0.167
[3,] 10 3 13 0.000

> ProbabilityForTriples(c(1,11,13),countmatrix)

Triple Order: 1 11 13 with estimated posterior probability = 0.785

[1,] 1 11 13 0.785
[2,] 1 13 11 0.215
[3,] 13 1 11 0.000

The estimated ordering of the triplets is correct for all cases when the drafts are spaced widely (i.e. no consecutive pair of drafts are in the triplet) and are estimated with a posterior probability of 79% or higher in each case. The farther the drafts are apart, the higher the estimated posterior probabilities, with probabilities reaching 1 when the drafts are spread far apart.

Finally, we show the sequencing results for consecutive triples of drafts:

> ProbabilityForTriples(c(10,11,12),countmatrix)

Triple Order: 10 11 12 with estimated posterior probability = 0.546

[1,] 10 11 12 0.546
[2,] 12 10 11 0.373
[3,] 10 12 11 0.081

> ProbabilityForTriples(c(5,6,7),countmatrix)
> ProbabilityForTriples(c(4,5,6), countmatrix)

Triple Order: 6 4 5 with estimated posterior probability = 0.538

[1,] 6 4 5 0.538
[2,] 4 5 6 0.462
[3,] 4 6 5 0.000

> ProbabilityForTriples(c(8,9,10), countmatrix)

Triple Order: 8 10 9 with estimated posterior probability = 0.607

[1,] 8 10 9 0.607
[2,] 8 9 10 0.393
[3,] 10 8 9 0.000

> ProbabilityForTriples(c(9,10,11), countmatrix)

Triple Order: 9 11 10 with estimated posterior probability = 0.447

[1,] 9 11 10 0.447
[2,] 9 10 11 0.362
[3,] 11 9 10 0.191

> ProbabilityForTriples(c(11,12,13), countmatrix)

Triple Order: 12 13 11 with estimated posterior probability = 0.500
As may be seen, triplets involving 3 consecutive drafts are also being sequenced correctly and for the most part with posterior probabilities higher than 50%. We note that the triplets (4, 5, 6), (8, 9, 10), (9, 10, 11), (11, 12, 13) are sequenced incorrectly. Specifically, drafts 4 and 5 have been interchanged in the estimation. This is most likely due to fact that versions 4 and 5 are very similar to one another when compared to all other drafts in the corpus (i.e. the distance between 4 and 5 is very small compared to other values in the distance matrix). So, the MCMC was not able to distinguish fully between the two versions. Also, triplets involving 9, 10, and 11 were sequenced incorrectly likely because the word length of 10 is greater than 11, breaking the monotonicity in the document lengths. Lastly, we observe that drafts 12 and 13 were interchanged in the estimated ordering. This can be explained by two possible reasonings: One is that the document length of 13 is smaller than that of 12. Another is that all of the earlier drafts (versions 1 through 8 inclusive) are closer in distance to draft 13 than they are to draft 12, causing the MCMC to order 13 before 12. All estimated probabilities for triplets surpass 1/3 by a considerable amount and so the results can be considered significant.

Overall, both the Take and Put One and MCMC methods did quite well in sequencing the drafts and subsets of the drafts for the Netflix data set.

### 6.1.2 “One Art” Poem Drafts

“One Art” is a famous poem written by Elizabeth Bishop. We found 14 drafts of this poem (Millier, 1990), which are believed to have been written over a period of two weeks. (There were originally 16 drafts that were believed to have been written, however, only 14 of them were available due to illegible handwriting in some of the drafts.) The final draft of this poem was published in 1976 and is displayed below:
ONE ART - by Elizabeth Bishop

The art of losing isn’t hard to master;
so many things seem filled with the intent
to be lost that their loss is no disaster.

Lose something every day. Accept the fluster
of lost door keys, the hour badly spent.
The art of losing isn’t hard to master.

Then practice losing farther, losing faster:
places, and names, and where it was you meant
to travel. None of these will bring disaster.

I lost my mother’s watch. And look! my last, or
next-to-last, of three loved houses went.
The art of losing isn’t hard to master.

I lost two cities, lovely ones. And, vaster,
some realms I owned, two rivers, a continent.
I miss them, but it wasn’t a disaster.

-Even losing you (the joking voice, a gesture
I love) I shan’t have lied. It’s evident
the art of losing’s not too hard to master
though it may look like (Write it!) like disaster.

Here again, the versions of this poem have been labelled according to their known true order and the
matrix of normalized Levenshtein distances between pairs of drafts was computed. The first version
is the longest and then the lengths vary without having any clear pattern relative to order. So the
pairwise Levenshtein distances are not monotone and we therefore would expect the sequencing
results using the Take and Put One algorithm to not be fully accurate.

The output shown below follows the same format as the previous example and includes:

(a) Lengths of the drafts, and the matrix of normalized Levenshtein distances between document pairs.

(b) Sequencing results using the Greedy Take and Put One algorithm with the SSE criterion.
(c) Results of the Metropolis-within-Gibbs algorithm with \( N = 1,000,000 \) iterations and a burn in period of \( B = 200,000 \), starting from the parameter values generated from the sequencing algorithm in (b). Once again, \( \alpha = 2.001 \) and \( \beta = 1.001\sigma^2 \) were selected as the values for the hyperparameters.

(d) 95\% and 99\% credible intervals for \( \sigma \) using the generated samples from (c).

(e) Results for sequencing triplets of documents for the three longest poems (1, 7, 9) and the three shortest poems (3, 4, 6). For each set of 3 documents, the estimated ordering (up to reversal) is shown along with its posterior probability. Also, displayed is a matrix containing each possible ordering of the 3 given elements and the proportion of samples amongst the 800,000 MCMC samples after burn-in that agreed with that particular ordering.

---

SEQUENCING RESULTS FOR "ONE ART" POEM DRAFTS
---

(a)

Document Lengths (presented in true ordering):

\[
\begin{array}{ccccccccccccccc}
\[,1\] & \[,2\] & \[,3\] & \[,4\] & \[,5\] & \[,6\] & \[,7\] & \[,8\] & \[,9\] & \[,10\] & \[,11\] & \[,12\] & \[,13\] & \[,14\] \\
297 & 87 & 60 & 26 & 90 & 27 & 288 & 220 & 249 & 212 & 173 & 185 & 179 & 153 \\
\end{array}
\]

Levenshtein Distance Matrix (presented in true ordering):

\[
\begin{array}{cccccccccccccccc}
\[,1\] & \[,2\] & \[,3\] & \[,4\] & \[,5\] & \[,6\] & \[,7\] & \[,8\] & \[,9\] & \[,10\] & \[,11\] & \[,12\] & \[,13\] & \[,14\] \\
[1,] & 0.0000 & 0.8175 & 0.8505 & 0.9304 & 0.7983 & 0.9100 & 0.7725 & 0.7191 & 0.7275 & 0.7125 & 0.7221 & 0.7197 & 0.7173 & 0.7323 \\
[2,] & 0.8175 & 0.0000 & 0.4223 & 0.7425 & 0.4020 & 0.7053 & 0.7773 & 0.7236 & 0.7417 & 0.7156 & 0.6546 & 0.6627 & 0.6590 & 0.6234 \\
[3,] & 0.8505 & 0.4223 & 0.0000 & 0.6531 & 0.4620 & 0.6062 & 0.8268 & 0.7647 & 0.7906 & 0.7601 & 0.7063 & 0.7206 & 0.7161 & 0.6741 \\
[4,] & 0.9304 & 0.7425 & 0.6531 & 0.0000 & 0.7820 & 0.5152 & 0.9276 & 0.9033 & 0.9155 & 0.9002 & 0.8769 & 0.8843 & 0.8806 & 0.8619 \\
[5,] & 0.7983 & 0.4020 & 0.4620 & 0.7820 & 0.0000 & 0.7360 & 0.7664 & 0.6978 & 0.7197 & 0.6971 & 0.6577 & 0.6382 & 0.6349 & 0.6163 \\
[6,] & 0.9100 & 0.7053 & 0.6062 & 0.5152 & 0.7360 & 0.0000 & 0.9028 & 0.8727 & 0.8850 & 0.8700 & 0.8407 & 0.8471 & 0.8455 & 0.8217 \\
[7,] & 0.7725 & 0.7773 & 0.8268 & 0.9276 & 0.7664 & 0.9028 & 0.0000 & 0.5944 & 0.6011 & 0.5932 & 0.6107 & 0.5932 & 0.5926 & 0.6126 \\
[8,] & 0.7191 & 0.7236 & 0.7647 & 0.9033 & 0.6978 & 0.8727 & 0.5944 & 0.0000 & 0.5245 & 0.5125 & 0.5181 & 0.5004 & 0.5197 & 0.5060 \\
[9,] & 0.7275 & 0.7417 & 0.7906 & 0.9155 & 0.7197 & 0.8850 & 0.6011 & 0.5245 & 0.0000 & 0.4642 & 0.5238 & 0.4911 & 0.5373 & 0.5586 \\
\end{array}
\]
(b)

====== RESULTS of Greedy TakeandPutOne for n= 14 using NNLS Optimization ======

Initial Order: 3 14 2 10 7 6 8 11 9 4 13 5 12 1
SSE: 16.2975

True Order: 1 2 3 4 5 6 7 8 9 10 11 12 13 14
SSE: 11.25073

Estimated Order: 4 6 3 2 5 14 13 12 11 10 8 9 7 1
SSE: 6.856161

Estimated Sigma = 0.2964785

Estimated Betas:
0.1044 0.2224 0.1103 0.0780 0.2840 0.0454 0.0383 0.0861 0.1113 0.0955 0.0845 0.1263 0.2152

Spearman Rank Correlation between True and Estimated Orderings: 0.1824
Euclidean Distance between True and Estimated Orderings: 372

The low Spearman correlation of 0.18 is somewhat misleading in this case as the estimated ordering has sequenced 7 through 14 in nearly perfect order (aside from the switching of versions 8 & 9) and has grouped together 2 through 6 although some elements have been misplaced in the grouping. In particular, 1 has been placed at the opposite end of where it should belong. The estimated ordering
is somewhat monotone increasing in the document lengths of the drafts, but not entirely.

(c)

====== RESULTS of METROPOLIS-WITHIN-GIBBS ALGORITHM ======

Number of iterations: \( N = 1e+06 \) Burn-in: \( B = 2e+05 \)

Using timeline/\(\sigma\) from Sequencing Algorithm as initial value

Initial value for timeline:

\[
\begin{array}{cccccccccccc}
0.104413 & 0.3268061 & 0.4370725 & 0.5150762 & 0.7990692 & 0.8445092 & 0.8827781 \\
0.9688523 & 1.080179 & 1.175688 & 1.2601500 & 1.386442 & 1.601681 \\
\end{array}
\]

Initial value for \(\sigma\): 0.2964785

Results based on 800,000 samples:

Estimated Ordering by MCMC:

4 6 3 2 5 14 13 12 11 10 8 9 7 1 with estimated posterior probability = 0.0005

Spearman Rank Correlation calculated between True and Estimated Orderings: 0.1824
Euclidean Distance between True and Estimated Orderings: 372

(d)

95% Credible Interval for \(\sigma\): (0.5380227, 0.7336044)
99% Credible Interval for \(\sigma\): (0.5146227, 0.774556)
In this case, the MCMC method estimated the same ordering as the Take and Put One algorithm. The posterior probability that the estimated ordering is the true one is 0.0005 which is significantly higher than $2/14!$. However, the relative belief ratio for the MCMC ordering is $21794573$, which is much lower than for the Netflix drafts, suggesting that the MCMC sequencing may not be as accurate as was the previous example. The credible interval estimates for $\sigma$ are small because the pairwise distances are normalized. Below are the sequencing results for the longest and the shortest three drafts of the poems:

(e)

<table>
<thead>
<tr>
<th>Triple Order</th>
<th>Estimated Posterior Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 7 9</td>
<td>0.618</td>
</tr>
<tr>
<td>1 9 7</td>
<td>0.270</td>
</tr>
<tr>
<td>9 1 7</td>
<td>0.112</td>
</tr>
</tbody>
</table>

The longest three drafts (1,7, and 9) are accurately sequenced by the MCMC algorithm with a posterior probability of 0.62, which is considerably higher than $1/3$. However, the shortest three drafts were incorrectly sequenced. The order of the triple is estimated as (3,6,4) with a posterior probability of 0.52, however the true order of (3,4,6) comes close with a probability of 0.43. It should be noted however that the three shortest drafts may not be entirely reliable as they may pertain only to parts of a draft (for example if some part of the original draft was missing and
therefore unavailable).

Overall, considering that monotonicity in the Levenshtein distances does not hold, the sequencing results for this poem can be considered to be adequate.

### 6.1.3 Versions of “The Song of Solomon”

“The Song of Solomon”, also known as “The Song of Songs”, is a love poem from the Old Testament and is believed to have been written by Solomon most likely around 965 B.C. The poem consists of 17 lines and was first included in the Hebrew Bible and later on in Christian versions of the Old Testament. We have found 10 versions of the poem from different translations of the Bible (such as Wycliffe, Knox, American Standard, Contemporary English, etc.), ranging over a period of 567 years. The approximate year in which each version was written is known via the publication date of the translation of the Bible in which it appears. It is of course possible that a version could have been written much before the publication date of the Bible in which it appears, but this should not cause any major discrepancies since the dates are all relatively spread apart (aside from versions 9 and 10, which were both published in 1995). In this data set, the documents are not actually drafts of the same work by the same author, but rather translations of the poem (mostly translated from Hebrew and ancient Greek into English) by different authors. The number of words in each version varies from 327 to 648, where the first version has the longest length but the lengths bear no clear pattern with the temporal order of the versions. Although they are not drafts, we can consider them to be similar and use Levenshtein distance between pairs of translations.

The output below shows the following:

(a) Lengths of the versions, year/date in which each version was written, and the matrix of normalized Levenshtein distances between document pairs.

(b) Sequencing results using the Greedy Take and Put One algorithm with the SSE criterion function.
(c) Results of the Metropolis-within-Gibbs algorithm with \( N = 100,000 \) iterations and a burn-in period of \( B = 20,000 \), starting from the parameter values generated from the sequencing algorithm in (b). \( \alpha = 2.001 \) and \( \beta = 1.001\sigma^2 \) were again selected as the values for the hyperparameters.

(d) 95% and 99% credible intervals for \( \sigma \) from (c).

------------

SEQUENCING RESULTS FOR VERSIONS OF THE "SONG OF SOLOMON"

------------

(a)

Document Lengths (presented in true ordering):

\[
\begin{array}{cccccccccc}
648 & 337 & 329 & 356 & 387 & 459 & 348 & 343 & 397 & 327 \\
\end{array}
\]

Dates (presented in true ordering):

\[
\begin{array}{cccccccccc}
\end{array}
\]

Distance Matrix (presented in true ordering):

\[
\begin{array}{cccccccccc}
[1,] & 0.0000 & 0.6353 & 0.6240 & 0.6447 & 0.6778 & 0.6654 & 0.6677 & 0.6418 & 0.6680 & 0.6958 \\
[2,] & 0.6353 & 0.0000 & 0.3902 & 0.3670 & 0.6552 & 0.5615 & 0.5237 & 0.2910 & 0.6004 & 0.6998 \\
[3,] & 0.6240 & 0.3902 & 0.0000 & 0.4263 & 0.6248 & 0.5770 & 0.5606 & 0.3859 & 0.6246 & 0.6931 \\
[4,] & 0.6447 & 0.3670 & 0.4263 & 0.0000 & 0.6557 & 0.4822 & 0.4774 & 0.2061 & 0.5473 & 0.7049 \\
[5,] & 0.6778 & 0.6552 & 0.6248 & 0.6557 & 0.0000 & 0.6615 & 0.6766 & 0.6547 & 0.7019 & 0.7046 \\
[6,] & 0.6654 & 0.5615 & 0.5770 & 0.4822 & 0.6615 & 0.0000 & 0.5239 & 0.4969 & 0.5921 & 0.6513 \\
[7,] & 0.6677 & 0.5237 & 0.5606 & 0.4774 & 0.6766 & 0.5239 & 0.0000 & 0.5068 & 0.4858 & 0.6368 \\
[8,] & 0.6418 & 0.2910 & 0.3859 & 0.2061 & 0.6547 & 0.4969 & 0.5068 & 0.0000 & 0.5599 & 0.6948 \\
\end{array}
\]
[9,] 0.6680 0.6004 0.6246 0.5473 0.7019 0.4858 0.5599 0.0000 0.6251
[10,] 0.6958 0.6998 0.6931 0.7049 0.7046 0.6513 0.6368 0.6948 0.6251 0.0000

(b)

========== RESULTS of Greedy TakeandPutOne for n = 10 using NNLS Optimization ========

Initial Order: 10 6 4 8 3 9 7 2 1 5
SSE: 3.405867

True Order: 1 2 3 4 5 6 7 8 9 10
SSE: 3.748835

Estimated Order: 5 10 9 7 4 8 2 3 6 1
SSE: 3.059421

Estimated Sigma = 0.29152

Estimated Betas:
0.1316 0.1946 0.1290 0.1720 0.0548 0.0743 0.0954 0.1377 0.2040

Spearman Rank Correlation between True and Estimated Orderings: -0.6364
Euclidean Distance between True and Estimated Orderings: 60

(c)

========== RESULTS of METROPOLIS-WITHIN-GIBBS ALGORITHM ========

Number of iterations: N = 1e+05 Burn-in: B = 20000

Using timeline/sigma from Sequencing Algorithm as initial value
Initial value for timeline:
0.1315853 0.3261711 0.45518 0.6271653 0.6819427 0.756291 0.8517148 0.989371 1.193331

Initial value for sigma: 0.29152

Results based on 80,000 samples:

Estimated Ordering:
5 10 9 7 8 4 3 2 6 1 with estimated posterior probability = 0.1214

Spearman Rank Correlation calculated by R: -0.7576
Euclidean Distance between True and Estimated Orderings: 50

(d)
95% Credible Interval for sigma: (22.11507, 34.41132)
99% Credible Interval for sigma: (20.88864, 37.31511)

We observe that the Take and Put One algorithm does reasonably well in sequencing with a Spearman correlation of -0.63 (and a negative value indicating that the estimated ordering is in reverse chronological order), especially considering that the distances are not monotone in ordering. The MCMC method further improves the estimated ordering with a Spearman correlation of -0.76. The relative belief ratio of the MCMC ordering is 220268.2. This estimated ordering has again no clear pattern with document length. The estimated ordering overall is quite good aside from versions 5 and 6 having been misplaced. Versions 1-4 are in their correct order and versions 7-10 are nearly in their proper order except that 7 and 8 are interchanged. We note here that versions 7 and 8 were written only 10 years apart from one another, which is considered to be close relative to other pairs of versions. This could explain why the MCMC algorithm switched them. We also observe that the 5th and 6th version were placed out of order in the estimated orderings both from
the Take and Put One algorithm as well as by the MCMC. One possible reason for this is that English versions of the Bible use different styles of translation (such as word-for-word, thought-for-thought, or paraphrasing) and so versions that use the same type of translation may be more alike than versions that do not use the same translation type, even if they were written farther apart in time. For example, we observe that translation 6 (Amplified Bible) is placed on the opposite end of translations 5, 9, and 10 (Knox, God’s Word and Contemporary English Version respectively). After further investigation into types of Bible translations, we found that the Amplified Bible is on one end of the spectrum using word-for-word translation, whereas Knox, God’s Word, and the Contemporary English versions are at the other end of the spectrum using a style in between thought-for-thought and paraphrasing. If the translation style for each version was available, then it could be used to perhaps improve the sequencing results. For example, we could group the versions by translation type, then sequence within each group, and finally sequence the groups together by using the “Fix T Take and Put One” algorithm (described in Section 4.5) several times. Each run of the Fix T Take and Put One algorithm would fix the positions of the elements that were sequenced within a particular group and the elements of another group would be placed relative to those fixed points. Nonetheless, the sequencing results from both the Take and Put One and MCMC methods obtained above are reasonably good.

The credible interval estimates for $\sigma$ seem to be high (compared to the estimate for the NNLS sequencing). One potential reason for the large estimates for $\sigma$ is that the MCMC is trying to compensate for the estimation of the $a_i$ points. The algorithm does not start off with the correct parameters for $a_i$ and hence it may tend to overestimate $\sigma$ so as to adjust for the estimation of $a_i$. A possible solution to this would be to use a narrower prior on $\sigma$, rather than a relatively flat one. It should also be noted that the credible intervals are in some sense not exactly estimating $\sigma$, the standard deviation parameter for the errors of the $Y_{ij}$, so the interpretation is not the same as would be for a Frequentist confidence interval. Rather, the MCMC method takes into account the prior distribution and is in fact estimating the mean of the posterior distribution of $\sigma$. Nonetheless, we would like the true value of $\sigma$ (which is unknown in this case) to be covered by the credible interval.
We mention here that poems are typically harder to accurately sequence than drafts of a manuscript since we would expect more substantial changes to occur between consecutive versions of a poem (so that versions written closer in time may actually be more dissimilar in their content). Also, changes may occur “back and forth” between the versions. For example, a poet may decide to remove an entire stanza or paragraph from their work and then decide to put it back in (perhaps with some minor edits) later on. This lack of monotonicity in the distances between versions will necessarily make the results less accurate, but the results were still quite reasonable for both sets of poems examined here. One possible way to improve the sequencing results would be to devise a variation of the Levenshtein distance that would consider a removal or insertion of an entire stanza/paragraph as being just one edit rather than counting single character edits. However, this distance would be rather involved to program and is outside the scope of this thesis.

6.2 Sequencing DEEDS Documents

The DEEDS (Documents of Early England Data Set) research project was founded by Michael Gervers, Professor of History, at the University of Toronto in 1975. Its main purpose was to create a database of medieval property transactions that record property transfer, both in terms of land, moveable goods, and rights, and then to extract pertinent information from the documents (such as names, places, relationships, properties, and dates if available) in order to date the undated charters. This would be of particular interest to historians as it would enable them to learn about the social, political, and economic situation of certain time periods. The DEEDS corpus contains over 10,000 charters written in Latin from the 7th to 14th century in England and Wales, consisting of approximately 60 different types of documents such as grants, wills, agreements, purchases, etc. Each charter contains the rights, obligations, and specific details about the exchange such as ownership, locations, and quantities. A typical charter contains approximately 200 words (or about 1300 characters). An example of a DEEDS manuscript is displayed in Appendix B. For further details, see Feuerverger et al (2008) and Tilahun (2011).

There were a total of 787 documents available for our use, comprising 28 different types of docu-
ments. Of these documents, there are 585 grants, and between 15-40 documents each of confirmation, extent of lands, wills, writs, and restitutions, for a total of 701 documents of these six most common types. The remaining types each have very few documents (between 1-5). Most of the charters were dated either internally or carefully assigned. Internally dated documents included calendar dates in the text itself or a major historical or astronomical event was referred to from which the date could be determined. On the other hand, documents without internal dates were assigned dates by historians on the basis of specific names and places that appeared in the text or by some other firm reason. We believe these estimated dates to be quite accurate and so for the purposes of these examples, we have labelled these estimated dates as being the true dates. In some cases, a range had been estimated for the date and in those cases we have used the midpoint of the minimum and maximum date as the true date. For the documents with internal or assigned dates, the dates range from year 604 to year 1072. The mean date is 902 with a standard deviation of approximately 108 years. The majority of the documents were written in the 10th century (see distribution in Figure 6.1 below). There are 24 documents amongst these that have not been assigned any date.

The document lengths vary between 18 and 2354 words and have no apparent relation with date or type of document. See Figures 6.2 and 6.3 below. The summary of descriptive statistics for document lengths is:

<table>
<thead>
<tr>
<th>Variable</th>
<th>n</th>
<th>Minimum</th>
<th>1st Quartile</th>
<th>Median</th>
<th>Mean</th>
<th>3rd Quartile</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document Length</td>
<td>787</td>
<td>18.0</td>
<td>151.5</td>
<td>284.0</td>
<td>322.4</td>
<td>435.5</td>
<td>2354.0</td>
</tr>
</tbody>
</table>

The entire corpus of 787 documents consists of a total of 37,899 unique words of which 24,291 (about 64%) occur in only one document and therefore are not useful for applying Latent Semantic Analysis procedures (see Section 2.3). Thus, there are a total of 13,608 unique words appearing in at least two documents in the corpus.
CHAPTER 6: SOME APPLICATIONS

Figure 6.1: Histogram of DEEDS documents dates. There are a total of 763 documents.

Figure 6.2: Plot of date vs. length for DEEDS documents. There are a total of 763 documents.

Figure 6.3: Boxplot of length by type for DEEDS documents with 6 most common types. There are a total of 701 documents.
In the examples of Sections 6.2.1, 6.2.2, 6.2.3, and 6.2.4 below, we will restrict our attention to the grant documents only. This is because the corpus predominantly consists of grants (74% are grants) and because we believe that sequencing within the same type should give better results than sequencing a corpus with different types (since documents of the same type will naturally have greater similarity and hence smaller pairwise distances than documents of different types).

Over the course of our experimentation with the DEEDS documents, we have made the following general observations:

1. Euclidean distances between document feature vectors that result from Latent Semantic Analysis and performing an SVD on a document-term matrix (as described in Section 2.3) provides the most useful distance measures between the documents for purposes of sequencing. When performing LSA, it is most effective to use frequency counts in the document-term matrix (as opposed to incidence counts) and to weight each entry by its tf-idf.

2. When available, it is better to use a training set of documents with known dates to sequence the documents in the test set using the Fix T Take and Put One algorithm from Section 4.5 rather than sequencing the documents in the test set alone using the Take and Put One algorithm from Section 4.1. The training set should be chosen so that it spans over the same domain as the test set and uses a similar vocabulary.

3. It is more effective to choose a larger number of documents in the training set than in the test set and to select the training set systematically by date order. Since the dates of the documents in the training set are known, this can be done, so that the documents in the training set are then relatively spread apart in time and thus allow for better sequencing results.

Since the above techniques improve sequencing results for the DEEDS data set, we will use them wherever possible in the following examples. We note that these observations seem reasonable to extend to other data sets as well and are in line with other findings in the text mining literature. A description of the DEEDS data format as well as the computer programs used for the following examples are given in Appendix B.
First, we present an example that compares sequencing using the SSE and the Average Distance to Peers (ADP) criterion functions in Equations 3.7.2 and 3.10.1 respectively.

### 6.2.1 Example: Comparing SSE and ADP Criterion Functions for Sequencing Grant Documents

In this example, we will use the same randomly selected set of \( n = 100 \) DEEDS documents to sequence via the Take and Put One algorithm (no training set is used) using both the SSE and the ADP criterion functions. The documents are labelled according to their true temporal order. Since the number of documents in the set is relatively large, we have chosen \( P = 10 \) peers for each document in the Closest Peers method, and a value of \( K = 10 \) for the rank of the SVD approximation. The output below displays the same details as the example from Section 4.3 except that sequencing is performed here on real documents rather than on simulated data:

---

DEEDS GRANT SEQUENCING RESULTS - ADP VS. SSE CRITERION FUNCTIONS

---

Number of Documents: \( n=100 \) & Number of Terms: \( m=2717 \)

Distances computed using:

- Euclidean Distances between Document Feature Vectors, where
- Frequency Counts are used in Document-Term Matrix
- Document-Term Matrix is preprocessed by tf-idf
- Rank of SVD Approximation: \( K=10 \)

====== RESULTS of Greedy TakeandPutOne for \( n=100 \) using Closest Peers Method ======

\( P=10 \) Peers for each element
CHAPTER 6: SOME APPLICATIONS

Initial Order: 89 76 11 90 87 62 65 73 39 82 81 48 53 86 21 51 26 25 63 43 97 55 49 79 29
59 38 64 95 30 36 69 99 78 44 77 60 33 74 96 71 4 68 5 47 50 93 45 98 32
13 57 34 24 18 58 27 41 56 75 31 91 85 37 1 15 19 61 72 3 70 14 66 9 20
84 100 35 67 54 22 2 6 88 52 80 7 94 23 8 92 10 42 46 17 16 83 28 12 40
ADP: 32.187

True Order: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
ADP: 17.937

Estimated Order: 94 50 40 64 76 99 97 85 75 82 86 78 96 93 63 60 90 89 80 84 67 47 42 73 79
46 58 91 98 77 74 88 71 95 62 68 51 65 61 59 53 87 45 83 92 69 56 72 52 39
34 100 70 81 33 66 27 26 3 35 32 38 4 20 55 49 9 17 24 14 37 5 6 31 28
22 13 12 15 21 18 7 25 2 11 19 36 1 8 30 48 44 54 57 29 41 23 16 43 10
ADP: 7.442

Number of Iterations: 427
Number of Permutations Tried: 34158

Spearman Rank Correlation between True and Estimated Orderings: -0.7374
Euclidean Distance between True and Estimated Orderings: 43766
Spearman correlation between distances and time differences: 0.3450
Euclidean Distance between True and Estimated Dates: 1640530

============= RESULTS of Greedy TakeandPutOne for n= 100 using NNLS Optimization =============

Initial Order: 98 54 89 67 10 61 50 76 9 4 17 52 58 72 3 51 91 57 99 45 69 64 35 34 32
7 92 84 1 73 15 59 86 55 16 53 63 41 97 87 83 93 2 26 71 75 70 40 88 42
14 94 18 29 44 30 90 48 24 49 31 80 28 95 78 60 19 37 25 20 56 46 23 11 43
33 22 85 47 21 74 100 13 81 79 62 6 68 38 77 82 5 27 65 96 39 12 66 8 36
CHAPTER 6: SOME APPLICATIONS

SSE: 303.3404

True Order: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
      26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
      51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
      76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
SSE: 267.8396

Estimated Order: 94 42 20 19 11 25 2 7 15 21 18 12 28 22 13 16 43 9 23 4 31 30 5 8 6
      29 41 55 14 24 37 1 57 44 54 48 36 66 35 26 32 27 38 3 34 52 100 39 70 72
      87 53 56 45 69 83 59 65 61 62 68 71 95 98 88 77 74 51 91 92 73 84 80 67 90
      89 60 63 93 78 96 82 86 75 58 85 97 99 76 79 64 50 40 46 10 81 33 49 47 17
SSE: 20.50734

Estimated Sigma= 0.06501882

Number of Iterations: 491
Number of Permutations Tried: 32718

Estimated Betas:

[1] 0.0752 0.2979 0.1403 0.0132 0.0774 0.0141 0.0720 0.0219 0.0025 0.0081 0.0275
[12] 0.0209 0.0025 0.0095 0.0236 0.0024 0.0042 0.0168 0.0027 0.0072 0.0061 0.0027
[23] 0.0031 0.0015 0.0029 0.0024 0.0021 0.0050 0.0013 0.0023 0.0029 0.0021 0.0022
[34] 0.0038 0.0015 0.0008 0.0025 0.0017 0.0004 0.0009 0.0020 0.0007 0.0027 0.0015
[45] 0.0042 0.0021 0.0011 0.0006 0.0026 0.0008 0.0010 0.0014 0.0011 0.0006 0.0019
[56] 0.0013 0.0010 0.0009 0.0007 0.0019 0.0003 0.0004 0.0012 0.0006 0.0004 0.0005
[67] 0.0014 0.0009 0.0009 0.0021 0.0035 0.0009 0.0006 0.0006 0.0009 0.0011 0.0008
[78] 0.0022 0.0036 0.0009 0.0022 0.0008 0.0022 0.0018 0.0026 0.0041 0.0082 0.0009
[89] 0.0092 0.0147 0.0108 0.0180 0.1413 0.0795 0.0481 0.0242 0.1035 0.1292 0.0206

Spearman Rank Correlation between True and Estimated Orderings: 0.6108
Euclidean Distance between True and Estimated Orderings: 64868
Spearman correlation between distances and time differences: 0.3450

Euclidean Distance between True and Estimated Dates: 2169289

Considering that there was no training set and that the documents were selected randomly (not systematically) so that the dates may not all be spread apart in time, both methods did reasonably well in sequencing the documents. The Closest Peers method did better with a Spearman correlation between the true and estimated orderings of approximately -0.74 compared to 0.61 for the NNLS Optimization. Both of these Spearman correlations are reasonable for a data set with $n = 100$ elements, especially considering that the pairwise distances between documents (obtained from LSA) and the time differences between documents have only 0.35 correlation. The estimated ordering from Closest Peers has a criterion value of $ADP = 7.442$, meaning that in this particular ordering, each document must step over an average of approximately 7 elements to reach each of its 10 most similar peers. For the most part, the Closest Peers method placed the later documents (documents 80-100) together in the estimated ordering. In both estimated orderings, the last document (100) is placed somewhere in the middle rather than at either end of the ordering and instead 94 is placed at the end. Also, in both cases, the criterion values for the estimated orderings is much lower than the values for the true ordering. This shows that the Take and Put One algorithm works at finding a minimum criterion function value (although it is not corresponding to the true ordering since the true ordering does not have the lowest value in this case). As expected, we see that the Closest Peers method sequences real documents more accurately than NNLS. These findings are consistent with many other examples that we tested.

In the next example, we demonstrate the Greedy Fix T Take and Put One algorithm of Section 4.5 with real documents.

6.2.2 Example: Sequencing Grant Documents with Training and Test Sets Chosen Systematically

Suppose we have a training or fixed set of $T$ documents in which the order is known and a test set of comparable documents which needs to be sequenced. As described in Section 4.5 the Greedy
Take and Put One algorithm can be used to sequence the entire set of documents. In order to
demonstrate the effectiveness of the Fix T Take and Put One, in this example both the fixed and
test sets were chosen systematically by date order as follows: Firstly, \( n = 100 \) documents were
randomly selected from the entire corpus and ordered by date and labelled according to their true
order. Since it improves results to select a larger fixed set when available, we chose to have \( T = 70 \)
documents in the fixed set and \( n - T = 30 \) documents for the test set. 30 of the 100 documents
were chosen systematically for the test set so that their dates are spread apart in time, allowing
for better sequencing. The remaining 70 documents were chosen to be in the fixed set. Below are
sequencing results for a collection of \( n = 100 \) documents (chosen as described above) with \( m = 2581 \)
unique terms that appear in at least two of the documents:

\[
\begin{array}{c}
\text{DEEDS GRANT SEQUENCING RESULTS USING TRAINING & TEST SETS} \\
\text{Number of Documents } n=100 \text{ and Number of Terms } m=2581 \\
\text{T=70 documents in training set chosen systematically and} \\
\text{n-T=30 in test set chosen systematically} \\
\text{Euclidean Distances between Document Feature Vectors computed, where} \\
\text{-Frequency Counts are used in Document-Term Matrix} \\
\text{-Document-Term Matrix is preprocessed by tf-idf} \\
\text{-Rank of SVD Approximation: } K=10 \\
\text{====== RESULTS of Greedy Fix T TakeandPutOne for T=70 & n-T=30 using NNLS Optimization ======} \\
\text{Document Numbers for } T=70 \text{ documents in Training Set:} \\
2 \ 3 \ 5 \ 6 \ 8 \ 9 \ 11 \ 12 \ 14 \ 15 \ 17 \ 18 \ 20 \ 21 \ 23 \ 24 \ 26 \ 27 \ 29 \ 30 \ 32 \ 33 \ 35 \ 36 \ 38 \ 39 \ 41 \ 42 \ 44 \ 45 \ 47 \ 48 \ 50 \ 51 \ 53 \ 54 \ 56 \ 57 \ 59 \ 60 \ 62 \ 63 \ 65 \ 66 \ 68 \ 69 \ 71 \ 72 \ 74 \ 75 \ 77 \ 78 \ 80 \ 81 \ 83 \ 84 \ 86 \ 87 \ 89 \ 90 \ 91 \ 92 \ 93 \ 94 \ 95 \ 96 \ 97 \ 98 \ 99 \ 100
\end{array}
\]
Document Numbers for the 30 documents in the Test Set:
1  4  7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88

Initial Order: (Order of training set fixed with test set interspersed randomly)
2  3  5  6  8  9 25 11 10 12 14 15 19 17 18 20 21 23 24 46 55 37 26 1  27
49 30 32 33 35 88 28 36 38 39 67 40 41 42 44 45 64 82 47 48 50 51 43 53
54 56 57 59 60 62 4 63 65 66 68 70 69 71 7 72 22 74 75 77 31 85 76 73
16 52 78 80 81 83 84 86 87 89 90 91 92 58 93 94 95 34 96 97 13 61 98 99 100
SSE: 548.8844

True Order: 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
SSE: 535.1297

Estimated Order: 19 34 10 43 28 13 2  3  5  6  8  9 11 12 14 15 17 18 20 21 23 24 26 27 29
30 32 33 35 16  7  1 36 38 39 41 42 44 45 47 48  4 25 50 52 31 37 22 51 55
40 61 88 58 53 54 56 57 59 46 60 62 63 65 70 66 68 69 71 72 74 75 77 78 80
81 83 64 84 73 85 76 49 86 87 89 90 91 92 93 94 95 96 97 98 99 79 100 67 82
SSE: 305.1064

Estimated Sigma= 0.2507899

Number of Iterations: 171
Number of Permutations Tried: 14115

Estimated Betas:

[1]  0.3312  0.0197  0.0462  0.0684  0.0857  0.0592  0.0000  0.0000  0.0000  0.0000  0.0000
[12]  0.0000  0.0000  0.0000  0.0000  0.0165  0.0000  0.0000  0.0000  0.0000  0.0347  0.0000
[23]  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0422  0.0104  0.0034  0.0207  0.0000
Estimated Ordering in Test Set:
19 34 10 43 28 13 16 7 1 4 25 52 31 37 22 55 40 61 88 58 46 70 64 73 85 76 49 79 67 82

True Ordering in Test Set:
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88

Spearman Rank Correlation between True and Estimated Orderings in Test Set: 0.8069
Euclidean Distance between True and Estimated Orderings in Test Set: 7812

Spearman Rank Correlation between True and Estimated Orderings in Entire Corpus: 0.9178
Euclidean Distance between True and Estimated Orderings in Entire Corpus: 13712

Spearman correlation between distances and time differences
between each pair of documents in the entire corpus: 0.2128
Euclidean Distance between all True and Estimated Dates: 319287

The sequencing was done quite accurately with a Spearman rank correlation of approximately 0.81 in the test set and 0.92 for the total ordering (for the ordering in the entire corpus). The distances and time differences between documents only have a Spearman correlation of 0.21 yet the sequencing is done reasonably well with NNLS optimization. There are many strings of documents that were sequenced or grouped in correct order such as document numbers 60-70 and 90-100. The later documents seem to be better sequenced than the earlier ones. Once again the SSE corresponding
to the estimated ordering is lower than that of the true ordering. Many of the estimated $\beta$ values are 0 indicating that the algorithm could not distinguish between some consecutive pairs of documents in the estimated ordering and also accounts for the switching of some elements. When examining several other examples like the above using the DEEDS corpus, we found that estimated orderings in the test set have a Spearman correlation of at least 0.70 with the true ordering.

In the next example, sequencing and MCMC procedures are performed using a training set and a test set.

### 6.2.3 Example: Sequencing and MCMC Inference for Grant Documents using a Training and a Test Set

First, 10 documents were randomly selected to be in the test set from the DEEDS grant corpus. From the remaining documents, $T = 20$ documents were chosen systematically so that the dates in the training set would be spread apart in time. This can often be done in practice to improve results. The test set may not have dates that are spread widely apart as they were chosen randomly. All documents have been labelled using their true temporal order. The output below shows the following details:

(a) Sequencing results using the Fix T Take and Put One algorithm with $ADP$ criterion function. A value of $P = 5$ is chosen for the number of peers for each document.

(b) Sequencing results using the Fix T Take and Put One algorithm with the $SSE$ criterion function. Both (a) and (b) use the same LSA-based distances between documents with $K = 5$ for the rank reduction.

(c) Results of the Metropolis-within-Gibbs algorithm with $N = 100,000$ iterations and a burn in period of $B = 20,000$, starting from the parameter values generated from the sequencing algorithm in (b). $\alpha = 2.001$ and $\beta = 1.001 \sigma^2$ were selected as the values for the hyperparameters.
(d) 95% and 99% credible intervals for $\sigma$ from (c).

DEEDS GRANTS SEQUENCING AND MCMC RESULTS USING A RANDOMLY CHOSEN TEST SET

Number of Documents $n=30$ and Number of Terms $m=793$

T=20 documents in training set chosen systematically

n-T=10 in test set chosen randomly

Document Numbers in Training Set (labelled according to true order):

1 3 4 5 7 8 12 13 14 15 17 19 20 22 23 24 25 27 28 29

Document Numbers for 10 documents in Test Set (labelled according to true order):

2 6 9 10 11 16 18 21 26 30

Euclidean Distances between Document Feature Vectors computed, where

- Frequency Counts are used in Document-Term Matrix
- Document-Term Matrix is preprocessed by tf-idf
- Rank of SVD Approximation: $K=5$

(a)

RESULTS of Greedy Fix T TakeandPutOne for T=20 and n-T=10 using Closest Peers Method

Initial Order: 1 3 4 26 16 10 21 5 18 6 7 8 12 13 30 14 15 17 19 20 11 9 22 23 24 25 27 2 28 29

ADP: 10.4

True Order: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

ADP: 7.5
Estimated Order: 6 1 2 11 3 4 10 9 5 7 8 12 13 14 18 15 17 19 20 22 23 24 25 27 28 29 21 26 30 16
ADP: 6.29

Number of Iterations: 24
Number of Permutations Tried: 786

Estimated Ordering in Test Set: 6 2 11 10 9 18 21 26 30 16
True Ordering in Test Set: 2 6 9 10 11 16 18 21 26 30

Spearman Rank Correlation between True and Estimated Orderings in Test Set: 0.8182
Euclidean Distance between True and Estimated Orderings in Test Set: 290

Spearman Rank Correlation between True and Estimated Total Orderings: 0.9066
(in entire corpus)
Euclidean Distance between True and Estimated Total Orderings: 420
(in entire corpus)

Spearman correlation between all pairwise distances and
time differences between documents: 0.115095

Euclidean Distance between all True and Estimated Dates: 62143.5

Dates of all Documents in True Order:
[1] 675.0 690.0 696.0 737.5 770.0 789.0 798.0 814.0 834.5 835.0
[11] 838.0 841.0 856.5 895.0 931.0 932.0 942.0 945.0 948.5 956.0
[21] 956.0 956.5 962.0 966.0 982.0 983.0 998.0 1018.0 1043.5 1046.0

Dates of all Documents in Estimated Order:
[1] 789.0 675.0 690.0 838.0 696.0 737.5 835.0 834.5 770.0 798.0
[11] 814.0 841.0 856.5 895.0 945.0 931.0 942.0 948.5 956.0 956.5
(b)

RESULTS of Greedy Fix T TakeandPutOne for T=20 and n-T=10 using NNLS Optimization

Initial Order: 18 16 1 10 3 4 5 7 6 2 8 26 12 13 14 15 11 17 19 22 20 21 22 24 25 27 30 21 28 29
SSE: 52.87303

True Order: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
SSE: 76.37363

Estimated Order: 30 6 1 3 4 5 7 8 12 10 13 14 11 2 9 18 15 17 19 20 26 21 22 23 24 25 26 27 28 29 16
SSE: 17.43013

Number of Iterations: 34
Number of Permutations Tried: 1417

Estimated Sigma= 0.2071989

Estimated Betas:

[1] 0.2060 0.1037 0.0000 0.0000 0.0677 0.0000 0.0000 0.0000 0.0323 0.0206

[11] 0.0000 0.0257 0.0025 0.0152 0.0075 0.0027 0.0013 0.0044 0.0000 0.0114

[21] 0.0034 0.0938 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 1.3828

Estimated Ordering in Test Set: 30 6 10 11 2 9 18 26 21 16
True Ordering in Test Set: 2 6 9 10 11 16 18 21 26 30

Spearman Rank Correlation between True and Estimated Orderings in Test Set: 0.2364

Euclidean Distance between True and Estimated Orderings in Test Set: 1162
Spearman Rank Correlation between True and Estimated Orderings: 0.7108
(in entire corpus)

Euclidean Distance between True and Estimated Orderings: 1300
(in entire corpus)

Spearman correlation between all distances and time differences: 0.1151
Euclidean Distance between all True and Estimated Dates: 220330

Dates of all Documents in True Order:

[1]  675.0  690.0  696.0  737.5  770.0  789.0  798.0  814.0  834.5  835.0  
[11] 838.0  841.0  856.5  895.0  931.0  942.0  945.0  948.5  956.0  
[21] 956.0  956.5  962.0  966.0  982.0  983.0  998.0  1018.0  1043.5  1046.0  

Dates of all Documents in Estimated Order:

[1]  1046.0  789.0  675.0  696.0  737.5  770.0  798.0  814.0  841.0  835.0  
[11] 856.5  895.0  838.0  690.0  834.5  945.0  931.0  942.0  948.5  956.0  
[21] 983.0  956.0  956.5  962.0  966.0  982.0  998.0  1018.0  1043.5  932.0  

(c)

======== RESULTS of METROPOLIS-WITHIN-GIBBS ALGORITHM ========

Number of iterations: N=1e+05 Burn-in: B=20000
Using timeline/σ from NNLS Sequencing Algorithm as initial value

Initial value for timeline:

0.2656932  0.2656932  0.2656932  0.2656932  0.2656932  0.3722283  0.3722283  0.3722283  
0.3722283  0.3722283  0.4501752  0.46175  0.4946992  0.50036  0.5038753  0.5129831  
0.5129831  0.5157113  0.5172785  0.5243284  0.558096  0.558096  0.558096  0.5974511  
0.5974511  0.5974511  0.5974511  0.7983909  1.0194
Initial value for sigma: 0.1837147

Estimated Ordering in Test Set:  2  6  9  10  11  16  18  21  26  30
with estimated posterior probability = 0.134

Spearman Rank Correlation between True and Estimated Orderings in Test set: 1
Euclidean Distance between True and Estimated Orderings in Test set: 0

95% Credible Interval for sigma: ( 0.1837147 ,  6.83952 )
99% Credible Interval for sigma: ( 0.1837147 ,  7.016302 )

We observe that Closest Peers (ADP criterion) performed much better than NNLS at sequencing this data set (Spearman correlations between true and ordering by Closest Peers is 0.82 (for test set only) and 0.91 (for all documents) and for NNLS only 0.24 (for test set only) and 0.71 (for all documents). Since the dates are not spread widely a part, Euclidean distances between the dates in the true and estimated orders are also given in the output. The Euclidean distance for Closest Peers (62143.5) is much less than that of the NNLS method (220,330). In this case, the MCMC provides a great improvement to the NNLS estimate of the ordering and in fact estimates the ordering perfectly even though the method used the values from the NNLS sequencing as initial values. The relative belief ratio for the MCMC ordering is 243129.6 in this case. In our experimentation, we have seen many examples like this in which the MCMC results in a far better estimate when the NNLS sequencing performs poorly. The credible interval estimates for $\sigma$ indicate that the estimated value of $\sigma$ from NNLS is low (this is because the estimated ordering from the Take and Put algorithm has a much lower SSE than the true ordering).

In the above examples, since the distances are not generated from the timeline points (unlike the examples of the previous chapters) and are instead calculated using features of the documents themselves, the scale for the $a_i$ and the pairwise distances between documents are different, and we would expect the distances to have a higher variance. For example, if the time differences between
versions are measured in months and we want to estimate how far apart in time documents \( i \) and \( j \) were written, we would have to take the interval estimate of \( a_i - a_j \) and divide by the appropriate scaling constant \( c \) (which is unknown in this case) to convert the time difference to months. Alternatively, rather than using Model 5.2.1, one could use the following model for the pairwise distances: \( Y_{ij} \sim N(c|a_j - a_i|, \sigma^2) \) and put a flat prior on \( c \). This alternative model is useful when one wishes to estimate the time using the exact same units of measurement as the original time differences between elements. However, this is not necessary for making general inferences based on MCMC (such as comparing time differences between pairs of elements) like we have done here and should anyway not affect the sequencing results.

In the next example, we compare all three sequencing methods proposed in this thesis for sequencing DEEDS grant documents using a training and test set.

6.2.4 Example: Comparing the Closest Peers, NNLS, and MCMC Methods for Sequencing DEEDS Grants

In this example, we present sequencing results using all three sequencing methods for 100 different samples of the DEEDS grant documents. First, 10 documents out of the 585 grants were randomly selected to be in the test set. From the remaining corpus, \( T = 20 \) documents were chosen systematically to be in the training set (so that the dates are spread relatively apart). This sampling procedure was repeated for \( M = 100 \) replications for a total of 100 different document samples of size 30 each. For each document sample, sequencing was performed using:

(a) Closest Peers Method: Fix \( T \) Take and Put One algorithm with \( ADP \) criterion function, where \( P = 5 \) peers was chosen for each document.

(b) Non-Negative Least Squares Method: Fix \( T \) Take and Put One algorithm with \( SSE \) criterion function.

(c) Markov Chain Monte Carlo Method: Metropolis-within-Gibbs algorithm with \( N = 100,000 \) iterations and a burn in period of \( B = 20,000 \). Initial values for \( a_i \) and \( \sigma \) were obtained using
the results from (b) with selected hyperparameter values of \( \alpha = 2.001 \) and \( \beta = 1.001\sigma^2 \).

All of the above methods use the same LSA-based distances between documents, namely Euclidean distances between document feature vectors resulting from a rank reduction with \( K = 5 \). For each sequencing method, the table below displays the following average results amongst the 100 document samples: criterion value (\( ADP, SSE, \) estimated posterior probability, \( \hat{p} \), or relative belief ratio \( RB \)), the absolute value of the Spearman correlation between the estimated and true orderings in the test set, and the Euclidean distance between the estimated and true orderings in the test set. The true ordering in the test set is denoted by \( \tau \) and the estimated orderings resulting from each sequencing method are subscripted with their respective method names:

<table>
<thead>
<tr>
<th>Method</th>
<th>Average Criterion Value</th>
<th>Average</th>
<th>Spearman Correlation</th>
<th>Average Euclidean Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) Closest Peers</td>
<td>( ADP(\hat{\tau}_{cp}) = 0.1861 )</td>
<td>0.8788</td>
<td></td>
<td>13.4164</td>
</tr>
<tr>
<td>(b) NNLS</td>
<td>( SSE(\hat{\tau}_{nnls}) = 22.1380 )</td>
<td>0.7576</td>
<td></td>
<td>18.9737</td>
</tr>
<tr>
<td>(c) MCMC</td>
<td>( \hat{p}(\hat{\tau}_{mcmc}) = 0.0804 )</td>
<td>0.8909</td>
<td></td>
<td>12.7279</td>
</tr>
<tr>
<td></td>
<td>( RB(\hat{\tau}_{mcmc}) = 145.877.8 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Truth</td>
<td>( ADP(\tau) = 0.4044 )</td>
<td>1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>( SSE(\tau) = 33.8006 )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.1: Sequencing Results in the Test Set for \( M = 100 \) document samples.

From the results in the table above, we observe that on average the NNLS and Closest Peers methods result in estimated orderings that actually have lower SSE or ADP criterion values than the true orderings. This phenomenon was also observed in several of the previous examples in this chapter. Although all methods do reasonably well, we note that on average, MCMC performs the best in sequencing, followed by Closest Peers, and lastly by NNLS. This can be seen by the Spearman correlation and Euclidean distance values between the estimated and true orderings (where higher correlations and smaller distances represent more accurate estimated orderings). The MCMC method results in a relative belief ratio of over 145,877 on average, which shows a very large increase in the belief that the MCMC ordering is the true one from a priori to a posteriori. On average, there is a 0.76 Spearman correlation between the true ordering and the
NNLS estimate, whereas there is a 0.88 correlation for the Closest Peers method. The MCMC method performs better than the others and as noted before, it can provide a great improvement when NNLS performs poorly (even when starting at the initial values obtained from NNLS).

6.3 A Method for Document Classification

We now turn our attention to a different but related problem for information retrieval applications. The focus of this thesis is sequencing documents and we note that there are more sophisticated methods used in the literature for classification problems (for instance, Damashek, 1995 and Rosenthal and Yoon, 2011). However, we present one method (with some variations) for classifying documents since we found that the tools we have developed for sequencing problems may be useful for classification and that tableaux distances (described in Section 2.5) may be effective for classification even though they were not particularly useful for sequencing.

Suppose then that a corpus has \( n \) documents, each of which belongs to one of \( c \) known categories/types/groups where the categories are labelled as 1, 2, \ldots, \( c \). Also, suppose that \( T \) of these \( n \) documents belong to a training set, \( T \), in which each document’s type is known and the remaining \( n - T \) documents comprise of a test set, \( V \), with documents that have not been classified but are known to belong to one of the \( c \) categories above. The goal is to classify each document in the test set into its correct category using the information in the training set. The method begins by placing each document in the training set into its respective category so that \( T \) can be partitioned into sets \( T_k \) consisting of \( T_k \) documents of type \( k \), for \( k = 1, 2, \ldots, c \) where \( T = \sum_{k=1}^{c} T_k \). Let \( D_i \) represent a document in \( V \) that needs to be classified. For each category \( k \), where \( k = 1, 2, \ldots, c \), the distance between \( D_i \) and each document \( D_j \in T_k \) is computed and is denoted as \( d(i, j) \) for \( j = 1, 2, \ldots, T_k \). (These distances can be tableaux, LSA-based, or any of the distance measures described in Chapter 2 or any other relevant distance measures.) The distance between \( D_i \) and a given category \( k \) is then calculated as the average distance between \( i \) and each of the documents in \( T_k \), i.e. \( d(i, T_k) = \frac{1}{T_k} \sum_{j \in T_k} d(i, j) \). The underlying idea is that a document should be on average closer to documents in its own category than to documents of other types. Using this premise, we
will make two attempts to estimate the type of each document in the test set as follows:
1) document $i$ will be classified as the type that has the minimal distance to $i$; and then
2) document $i$ will be classified as the type that has the second smallest distance to $i$.

Alternatively, when classifying document $i$, one could take into account information only from the documents in the training set that are the closest to $i$ rather than considering the entire training set (similar to the Closest Peers method from Section 3.10). Thus, instead of averaging all the distances in each category, one would average the distances of only the $P$ closest peers to $i$ in each category, i.e. the distance from $i$ to each category $k$ is computed as $d(i, T_k) = \frac{1}{P} \sum_{j \in P_i(k)} d(i, j)$ for $k = 1, 2, \ldots, c$. Here, the set $P_i(k)$ denotes the $P$ closest peers to $i$ that are of type $k$ in the training set, where the number of peers $P$ is selected according to the total number of elements. Values of $P = 5, 10$ are typically used. Then, document $i$ is classified using two attempts in the same manner as above.

This classification method can be applied to document collections where there are inherent groupings that can be detected by word patterns in the corpus (such as subject and language identification) and could possibly be extended to other problems such as spam filtering and authorship detection. Different types of distance measures (see Chapter 2) can be utilized depending on the type of classification problem.

Below we present some examples to demonstrate the application of this classification method for the DEEDS corpus.

### 6.3.1 Classifying DEEDS Documents

The DEEDS document set that is used in this thesis consists of predominantly 6 types of documents with the number of documents in each type displayed in parentheses: grant (585), confirmation (42), extent of lands (31), will (22), restitution (18), and writ (17). Since all the other types occur scarcely in the corpus, they have been omitted in the following examples so that only the above 6 types will be used. See Section 6.2 for a more complete description of this data set. Since there
are significantly many more grants in the corpus than any other type, in order to prevent the results from becoming skewed and to reduce computation, only some of the grants are used. Note that when there are many documents, the computational intensity is considerably higher for the Tableaux method since common shingles of different sizes need to be considered in order to create a tableau for each pair of documents, however it is only moderately intense when using LSA-based or other distance measures between documents. Altogether 80 grants were randomly chosen from the grant corpus, and all documents in the remaining 5 types were used for the examples below.

6.3.2 Example: Classifying DEEDS Documents using Tableaux Distances

As described above, 80 grants were randomly selected for use in this example and all documents from the remaining 5 types were used resulting in a total of $n = 210$ documents. $T = 160$ of these documents were randomly selected to be in the training set and the remaining $n - T = 50$ were considered as the test set. Tableaux similarities between documents were computed using the parameter values of $\alpha = 2$, $\beta = 1$, $\gamma = 0.5$ in Equation 2.5.3. These similarities were converted to distances using method (2) from Section 2.6 and then classification was performed using the procedure outlined in Section 6.3. Since the types are known for all documents (including those in the test set), the accuracy of the classification method can be verified. For each document in the test set, the output below shows the document number (the number within the entire corpus), the document’s true type, the classified type on the first attempt, and classified type on the second attempt:

---

**TYPE CLASSIFICATION OF DEEDS DOCUMENTS using Tableaux Distances**

---

Number of Documents in Full Corpus: $n=210$  
Number of Terms: $m=4794$  
Number of documents in Training Set: $T=160$  
Number of documents to classify (in Test Set): $n-T=50$
Number of Documents of each Type in Entire Corpus:

"Grant" "Confirmation" "Extent of Lands" "Will" "Restitution" "Writ"

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grant</td>
<td>80</td>
</tr>
<tr>
<td>Confirmation</td>
<td>42</td>
</tr>
<tr>
<td>Extent of Lands</td>
<td>31</td>
</tr>
<tr>
<td>Will</td>
<td>22</td>
</tr>
<tr>
<td>Restitution</td>
<td>18</td>
</tr>
<tr>
<td>Writ</td>
<td>17</td>
</tr>
</tbody>
</table>

Tableaux Parameters: $\alpha = 2$, $\beta = 1$, $\gamma = 0.5$

Classification Results:

<table>
<thead>
<tr>
<th>Document Number</th>
<th>True Type</th>
<th>Classified (Estimated) Type 1</th>
<th>Classified (Estimated) Type 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: 3</td>
<td>Grant</td>
<td>Grant</td>
<td>Confirmation</td>
</tr>
<tr>
<td>2: 5</td>
<td>Grant</td>
<td>Grant</td>
<td>Confirmation</td>
</tr>
<tr>
<td>3: 6</td>
<td>Grant</td>
<td>Grant</td>
<td>Confirmation</td>
</tr>
<tr>
<td>4: 12</td>
<td>Grant</td>
<td>Grant</td>
<td>Restitution</td>
</tr>
<tr>
<td>5: 22</td>
<td>Grant</td>
<td>Grant</td>
<td>Confirmation</td>
</tr>
<tr>
<td>6: 29</td>
<td>Grant</td>
<td>Grant</td>
<td>Confirmation</td>
</tr>
<tr>
<td>7: 30</td>
<td>Grant</td>
<td>Grant</td>
<td>Confirmation</td>
</tr>
<tr>
<td>8: 31</td>
<td>Grant</td>
<td>Extent of Lands</td>
<td>Grant</td>
</tr>
<tr>
<td>9: 34</td>
<td>Grant</td>
<td>Grant</td>
<td>Confirmation</td>
</tr>
<tr>
<td>10: 38</td>
<td>Grant</td>
<td>Grant</td>
<td>Will</td>
</tr>
<tr>
<td>11: 45</td>
<td>Grant</td>
<td>Grant</td>
<td>Restitution</td>
</tr>
<tr>
<td>12: 49</td>
<td>Grant</td>
<td>Grant</td>
<td>Restitution</td>
</tr>
<tr>
<td>13: 53</td>
<td>Grant</td>
<td>Grant</td>
<td>Restitution</td>
</tr>
<tr>
<td>14: 56</td>
<td>Grant</td>
<td>Grant</td>
<td>Confirmation</td>
</tr>
<tr>
<td>15: 65</td>
<td>Grant</td>
<td>Grant</td>
<td>Confirmation</td>
</tr>
<tr>
<td>16: 66</td>
<td>Grant</td>
<td>Grant</td>
<td>Extent of Lands</td>
</tr>
<tr>
<td>17: 71</td>
<td>Grant</td>
<td>Grant</td>
<td>Restitution</td>
</tr>
<tr>
<td>18: 72</td>
<td>Grant</td>
<td>Grant</td>
<td>Restitution</td>
</tr>
<tr>
<td>19: 74</td>
<td>Grant</td>
<td>Grant</td>
<td>Confirmation</td>
</tr>
<tr>
<td>20: 75</td>
<td>Grant</td>
<td>Grant</td>
<td>Restitution</td>
</tr>
<tr>
<td>21: 78</td>
<td>Grant</td>
<td>Grant</td>
<td>Confirmation</td>
</tr>
<tr>
<td>22: 79</td>
<td>Grant</td>
<td>Grant</td>
<td>Restitution</td>
</tr>
<tr>
<td>23: 81</td>
<td>Confirmation</td>
<td>Will</td>
<td>Confirmation</td>
</tr>
<tr>
<td>24: 84</td>
<td>Confirmation</td>
<td>Restitution</td>
<td>Confirmation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>25: 101</td>
<td>Confirmation</td>
<td>Confirmation</td>
<td>Restitution</td>
</tr>
<tr>
<td>26: 104</td>
<td>Confirmation</td>
<td>Grant</td>
<td>Confirmation</td>
</tr>
<tr>
<td>27: 112</td>
<td>Confirmation</td>
<td>Confirmation</td>
<td>Restitution</td>
</tr>
<tr>
<td>28: 118</td>
<td>Confirmation</td>
<td>Grant</td>
<td>Confirmation</td>
</tr>
<tr>
<td>29: 119</td>
<td>Confirmation</td>
<td>Grant</td>
<td>Extent of Lands</td>
</tr>
<tr>
<td>30: 120</td>
<td>Confirmation</td>
<td>Grant</td>
<td>Confirmation</td>
</tr>
<tr>
<td>31: 121</td>
<td>Confirmation</td>
<td>Confirmation</td>
<td>Grant</td>
</tr>
<tr>
<td>32: 123</td>
<td>Extent of Lands</td>
<td>Grant</td>
<td>Restitution</td>
</tr>
<tr>
<td>33: 132</td>
<td>Extent of Lands</td>
<td>Extent of Lands</td>
<td>Confirmation</td>
</tr>
<tr>
<td>34: 139</td>
<td>Extent of Lands</td>
<td>Extent of Lands</td>
<td>Confirmation</td>
</tr>
<tr>
<td>35: 141</td>
<td>Extent of Lands</td>
<td>Grant</td>
<td>Confirmation</td>
</tr>
<tr>
<td>36: 145</td>
<td>Extent of Lands</td>
<td>Extent of Lands</td>
<td>Grant</td>
</tr>
<tr>
<td>37: 146</td>
<td>Extent of Lands</td>
<td>Writ</td>
<td>Will</td>
</tr>
<tr>
<td>38: 160</td>
<td>Will</td>
<td>Will</td>
<td>Writ</td>
</tr>
<tr>
<td>39: 161</td>
<td>Will</td>
<td>Will</td>
<td>Writ</td>
</tr>
<tr>
<td>40: 163</td>
<td>Will</td>
<td>Will</td>
<td>Writ</td>
</tr>
<tr>
<td>41: 169</td>
<td>Will</td>
<td>Will</td>
<td>Writ</td>
</tr>
<tr>
<td>42: 175</td>
<td>Will</td>
<td>Will</td>
<td>Writ</td>
</tr>
<tr>
<td>43: 176</td>
<td>Restitution</td>
<td>Restitution</td>
<td>Grant</td>
</tr>
<tr>
<td>44: 179</td>
<td>Restitution</td>
<td>Restitution</td>
<td>Confirmation</td>
</tr>
<tr>
<td>45: 187</td>
<td>Restitution</td>
<td>Grant</td>
<td>Confirmation</td>
</tr>
<tr>
<td>46: 189</td>
<td>Restitution</td>
<td>Grant</td>
<td>Restitution</td>
</tr>
<tr>
<td>47: 190</td>
<td>Restitution</td>
<td>Grant</td>
<td>Confirmation</td>
</tr>
<tr>
<td>48: 199</td>
<td>Writ</td>
<td>Confirmation</td>
<td>Writ</td>
</tr>
<tr>
<td>49: 201</td>
<td>Writ</td>
<td>Writ</td>
<td>Will</td>
</tr>
<tr>
<td>50: 203</td>
<td>Writ</td>
<td>Writ</td>
<td>Will</td>
</tr>
</tbody>
</table>

Proportion of times type is classified correctly on 1st attempt: 0.72
Proportion of times type is classified correctly on 2nd attempt: 0.16
Proportion of times type is classified correctly on either attempt: 0.88

Table 6.2 further below summarizes these results.
### Classification Results for DEEDS Documents using Tableaux Distances

<table>
<thead>
<tr>
<th>True Type</th>
<th>1st Classification</th>
<th>2nd Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grant (22)</td>
<td>Grant (21)</td>
<td>Grant (1)</td>
</tr>
<tr>
<td></td>
<td>Confirmation (0)</td>
<td>Confirmation (11)</td>
</tr>
<tr>
<td></td>
<td>Extent of Lands (1)</td>
<td>Extent of Lands (1)</td>
</tr>
<tr>
<td></td>
<td>Will (0)</td>
<td>Will (1)</td>
</tr>
<tr>
<td></td>
<td>Restitution (0)</td>
<td>Restitution (8)</td>
</tr>
<tr>
<td></td>
<td>Writ (0)</td>
<td>Writ (0)</td>
</tr>
<tr>
<td>Confirmation (9)</td>
<td>Grant (4)</td>
<td>Grant (1)</td>
</tr>
<tr>
<td></td>
<td>Confirmation (3)</td>
<td>Confirmation (5)</td>
</tr>
<tr>
<td></td>
<td>Extent of Lands (0)</td>
<td>Extent of Lands (1)</td>
</tr>
<tr>
<td></td>
<td>Will (1)</td>
<td>Will (0)</td>
</tr>
<tr>
<td></td>
<td>Restitution (1)</td>
<td>Restitution (2)</td>
</tr>
<tr>
<td></td>
<td>Writ (0)</td>
<td>Writ (0)</td>
</tr>
<tr>
<td>Extent of Lands (6)</td>
<td>Grant (2)</td>
<td>Grant (1)</td>
</tr>
<tr>
<td></td>
<td>Confirmation (0)</td>
<td>Confirmation (3)</td>
</tr>
<tr>
<td></td>
<td>Extent of Lands (3)</td>
<td>Extent of Lands (0)</td>
</tr>
<tr>
<td></td>
<td>Will (0)</td>
<td>Will (1)</td>
</tr>
<tr>
<td></td>
<td>Restitution (0)</td>
<td>Restitution (1)</td>
</tr>
<tr>
<td></td>
<td>Writ (1)</td>
<td>Writ (0)</td>
</tr>
<tr>
<td>Will (5)</td>
<td>Grant (0)</td>
<td>Grant (0)</td>
</tr>
<tr>
<td></td>
<td>Confirmation (0)</td>
<td>Confirmation (0)</td>
</tr>
<tr>
<td></td>
<td>Extent of Lands (0)</td>
<td>Extent of Lands (0)</td>
</tr>
<tr>
<td></td>
<td>Will (5)</td>
<td>Will (0)</td>
</tr>
<tr>
<td></td>
<td>Restitution (0)</td>
<td>Restitution (0)</td>
</tr>
<tr>
<td></td>
<td>Writ (0)</td>
<td>Writ (5)</td>
</tr>
<tr>
<td>Restitution (5)</td>
<td>Grant (3)</td>
<td>Grant (1)</td>
</tr>
<tr>
<td></td>
<td>Confirmation (0)</td>
<td>Confirmation (3)</td>
</tr>
<tr>
<td></td>
<td>Extent of Lands (0)</td>
<td>Extent of Lands (0)</td>
</tr>
<tr>
<td></td>
<td>Will (0)</td>
<td>Will (0)</td>
</tr>
<tr>
<td></td>
<td>Restitution (2)</td>
<td>Restitution (1)</td>
</tr>
<tr>
<td></td>
<td>Writ (0)</td>
<td>Writ (0)</td>
</tr>
<tr>
<td>Writ (3)</td>
<td>Grant (0)</td>
<td>Grant (0)</td>
</tr>
<tr>
<td></td>
<td>Confirmation (1)</td>
<td>Confirmation (0)</td>
</tr>
<tr>
<td></td>
<td>Extent of Lands (0)</td>
<td>Extent of Lands (0)</td>
</tr>
<tr>
<td></td>
<td>Will (0)</td>
<td>Will (2)</td>
</tr>
<tr>
<td></td>
<td>Restitution (0)</td>
<td>Restitution (0)</td>
</tr>
<tr>
<td></td>
<td>Writ (2)</td>
<td>Writ (1)</td>
</tr>
</tbody>
</table>

Table 6.2: DEEDS Classification Results using Tableaux Distances for example from Section 6.3.2

The number of documents in the training set and how many of them were classified as each type on each classification attempt are displayed in parentheses beside the type name.
The results show that 21 out of the 22 grants in the test set (95%) were correctly classified on the 1st attempt and all of them were classified correctly by either of the attempts. All the wills were classified properly on the first attempt and confirmation type documents were classified better on the 2nd attempt rather than the first, with an overall success rate of 8/9. Restitutions did not get classified as well as other types and writs were classified by either attempt correctly 100% of the time. The documents in the test set were classified correctly 72% of the time on the first attempt and 88% of the time by either of the attempts. These results can be considered significant as the proportions are much higher than \( \approx \frac{1}{6} \), the proportion of times each type would be estimated under pure randomness. It can also be seen that confirmations, restitutions, and grants are more similar to one another than to other types, and that writs and wills are also similar in type.

Overall, the classification results are good when using tableaux distances. In the next example, we illustrate the classification method using LSA-based distances.

### 6.3.3 Example: Classifying DEEDS Documents using LSA-based Distances

For this example, we have used the exact same setup as the previous one so that we may compare the results. As before, there are a total of \( n = 210 \) documents consisting of 6 types. The same \( T = 160 \) documents and \( n - T = 50 \) documents were selected to be in the training and test sets respectively as were chosen in the previous example. Euclidean distances between feature vectors resulting from performing LSA were calculated between a given document in the test set and each document in the training set. The type of each document in the test set was estimated following the procedure described in Section 6.3 by two methods: (1) averaging all the distances, and (2) by averaging only the closest \( P = 10 \) peers of each type. For each document in the test set, the output below displays the document number, the document’s true type, the classified type on the first attempt, and classified type on the second attempt for both methods:

-------------------

**TYPE CLASSIFICATION OF DEEDS DOCUMENTS using LSA-based Distances:**

-------------------
Number of Documents in Full Corpus: \( n=210 \) and Number of Terms: \( m=4794 \)

Number of documents in Training Set: \( T=160 \)

Number of documents to classify (in Test set): \( n-T=50 \)

Number of Documents of each Type in Entire Corpus:

"Grant" "Confirmation" "Extent of Lands" "Will" "Restitution" "Writ"

\[
\begin{array}{cccccc}
80 & 42 & 31 & 22 & 18 & 17
\end{array}
\]

Euclidean Distances between Document Feature Vectors computed, where

- Frequency Counts are used in Document-Term Matrix
- Document-Term Matrix is preprocessed by tf-idf
- Rank of SVD Approximation: \( K=15 \)

Classification performed comparing:

Averaging All Distances VS. Averaging Distances of Closest \( P=10 \) Peers

Classification Results:

<table>
<thead>
<tr>
<th>Document Number</th>
<th>True Type</th>
<th>AVERAGING ALL DISTANCES</th>
<th>AVERAGING CLOSEST PEERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: 3 Grant</td>
<td>Grant</td>
<td>Restitution</td>
<td>Grant Confirmation</td>
</tr>
<tr>
<td>2: 5 Grant</td>
<td>Grant</td>
<td>Restitution</td>
<td>Grant Confirmation</td>
</tr>
<tr>
<td>3: 6 Grant</td>
<td>Grant</td>
<td>Restitution</td>
<td>Grant Confirmation</td>
</tr>
<tr>
<td>4: 12 Grant</td>
<td>Grant</td>
<td>Restitution</td>
<td>Grant Confirmation</td>
</tr>
<tr>
<td>5: 22 Grant</td>
<td>Grant</td>
<td>Restitution</td>
<td>Grant Confirmation</td>
</tr>
<tr>
<td>6: 29 Grant</td>
<td>Grant</td>
<td>Restitution</td>
<td>Grant Confirmation</td>
</tr>
<tr>
<td>7: 30 Grant</td>
<td>Grant</td>
<td>Restitution</td>
<td>Grant Confirmation</td>
</tr>
<tr>
<td>8: 31 Grant</td>
<td>Grant</td>
<td>Restitution</td>
<td>Grant Confirmation</td>
</tr>
</tbody>
</table>
9:  34 Grant | Grant Restitution | Grant Confirmation
10: 38 Grant | Grant Restitution | Grant Confirmation
11: 45 Grant | Restitution Grant | Grant Confirmation
12: 49 Grant | Grant Restitution | Grant Confirmation
13: 53 Grant | Grant Restitution | Grant Confirmation
14: 56 Grant | Grant Restitution | Grant Confirmation
15: 65 Grant | Extent of Lands Restitution | Extent of Lands Confirmation
16: 66 Grant | Restitution Grant | Confirmation Grant
17: 71 Grant | Restitution Grant | Grant Confirmation
18: 72 Grant | Restitution Grant | Restitution Grant
19: 74 Grant | Grant Restitution | Grant Confirmation
20: 75 Grant | Grant Restitution | Grant Confirmation
21: 78 Grant | Grant Restitution | Grant Confirmation
22: 79 Grant | Restitution Grant | Grant Confirmation
23: 81 Confirmation | Grant Restitution | Confirmation Grant
24: 84 Confirmation | Restitution Grant | Confirmation Grant
25: 101 Confirmation | Restitution Grant | Grant Confirmation
26: 104 Confirmation | Grant Restitution | Grant Confirmation
27: 112 Confirmation | Grant Restitution | Will Confirmation
28: 118 Confirmation | Grant Restitution | Grant Confirmation
29: 119 Confirmation | Grant Restitution | Grant Confirmation
30: 120 Confirmation | Grant Restitution | Grant Confirmation
31: 121 Confirmation | Grant Restitution | Grant Confirmation
32: 123 Extent of Lands | Grant Restitution | Grant Confirmation
33: 132 Extent of Lands | Extent of Lands Restitution | Extent of Lands Confirmation
34: 139 Extent of Lands | Extent of Lands Restitution | Extent of Lands Confirmation
35: 141 Extent of Lands | Grant Restitution | Grant Confirmation
36: 145 Extent of Lands | Grant Restitution | Extent of Lands Grant
37: 146 Extent of Lands | Grant Restitution | Grant Confirmation
38: 160 Will | Grant Restitution | Will Confirmation
39: 161 Will | Grant Restitution | Will Confirmation
40: 163 Will | Grant Restitution | Will Confirmation
41: 169 Will | Grant Restitution | Will Confirmation
42: 175 Will | Grant Restitution | Will Grant
43: 176 Restitution | Restitution Grant | Grant Confirmation
The above results show that averaging only peers provides an improvement over averaging all distances when classifying documents. Overall, the closest peers method of classification estimated the type correctly 80% of the time, whereas the method that averages all distances classified accurately only 60% of the time. There are only very few cases when using all distances correctly estimates the type and closest peers does not (10% all corresponding to restitutions). Another notable observation is that the closest peers method classifies all the wills correctly on the first attempt, whereas the other method does not accurately classify any of them (on the first or second attempt). Similarly, none of the confirmations are accurately classified when all distances are averaged, but they are all accurately classified (on either attempt) using closest peers. Once again, grants, confirmation, and restitution documents seem to appear quite similar to one another in both methods.

The results of the above two examples are quite typical for the DEEDS data set. In our experimentation, we have seen that tableaux distances are more effective in classifying documents than are LSA-based distances, although they both work quite well. In particular, averaging closest peers gives more accurate results than considering all distances. The classification method us-
ing tableaux distances correctly classifies documents at least 70% of the time on the first attempt whereas LSA-based distances are effective at least 60% of the time. Overall, both types of distances work reasonably well for classification purposes.

### 6.4 Summary of Results for Real Data Sets

Below we present a short summary of the sequencing and classification results obtained for the real data sets used in this thesis. In particular, for each data set, we identify which types of distances work best and which methods result in an estimated ordering that is the closest to the true ordering of the documents.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Type of Distance Measure</th>
<th>Sequencing/Classification Method</th>
<th>Page References</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SEQUENCING:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>Drafts / Versions of a Work:</em></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Netflix Paper</td>
<td>Normalized</td>
<td>MCMC and NNLS</td>
<td>99-108</td>
</tr>
<tr>
<td>One Art Poem</td>
<td>Levenshtein Distance</td>
<td>MCMC and NNLS</td>
<td>108-114</td>
</tr>
<tr>
<td>Song of Solomon</td>
<td></td>
<td>MCMC</td>
<td>114-119</td>
</tr>
<tr>
<td><em>DEEDS:</em></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Grant Corpus (without using a training set)</td>
<td>Euclidean Distance between Document Feature Vectors (when Document-Term Matrix weighted by tf-idf in LSA)</td>
<td>Closest Peers</td>
<td>123-126</td>
</tr>
<tr>
<td>Grant Corpus (using training set)</td>
<td></td>
<td>MCMC</td>
<td>130-136</td>
</tr>
<tr>
<td><strong>CLASSIFICATION:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>DEEDS</em></td>
<td>Tableaux Distances</td>
<td>Averaging Distances of Closest Peers</td>
<td>140-148</td>
</tr>
</tbody>
</table>

In this chapter, we have presented some real examples involving various collections of documents including different types, languages, and time periods over which the documents span. Our sequencing and other inference procedures worked reasonably well for all collections. All methods were effective for sequencing drafts without using a training set. The Closest Peers method was superior to NNLS for sequencing real documents (as opposed to simulated numbers with distances such
as in the examples from Chapters 3 and 4, although both methods worked reasonably. Overall, MCMC is the most advantageous method as it sequences documents and drafts the most accurately, can provide improvement when NNLS performs poorly (even when starting at the initial values obtained from NNLS), and also allows for making inferences about subsets of documents. We note that all the above methods can be implemented without the use of a training set, however results will be improved by using a training set that has been chosen systematically. Tableaux distances were most effective for classifying documents by type, whereas LSA-based distances perform better for sequencing documents and Levenshtein distance was most appropriate for sequencing drafts. In particular, averaging just the closest peers (rather than using all the distances) generally provides an improvement for classification.
Chapter 7

Complements and Concluding Remarks

We begin by commenting that the research problem of sequencing documents is an inherently difficult one but very interesting to pursue. This thesis has covered a wide range of methods for document sequencing problems and various related topics. A summary of these methods with possible improvements are given in Section 7.1, future work and extensions to our work are discussed in Section 7.2 and further potential applications are suggested in Section 7.3.

7.1 Summary of Methods and Possible Improvements

Below is a summary of the methods that were detailed in our work:

- Various distance measures between documents were proposed in Chapter 2, including measures that use direct word frequencies, LSA-based distances, and new types of distances such as Tableaux-based distances. We also outlined different approaches for attempting to identify which types of distance measures are most aligned with time separation (i.e. which have a high degree of correlation with time difference between documents).
• Theory was developed for a sequencing method based on a certain type of regression model on the pairwise distances that uses NNLS optimization (in Section 3.3) and the Closest Peers method which examines the proximity of each element to its most similar peers was also proposed (in Section 3.10). These methods led to two different criteria that are functions of an ordering that can be used for sequencing: Error Sum of Squares (SSE) and Average Distance to Peers (ADP). We demonstrated that both criterion functions performed well in sequencing, but that ADP surpassed SSE when sequencing real text documents.

• The Greedy Take and Put One algorithm was devised in Section 4.1 to efficiently move between permutations in order to optimize a given criterion function. Variations of this optimization algorithm were also discussed and of particular utility was the Fix T Take and Put One (described in Section 4.5) that can be applied to document collections that have an appropriate training/fixed set and test set.

• A Markov Chain Monte Carlo approach was developed in Chapter 5 to sequence documents, estimate timeline positions, and make various types of inferences about subsets of documents. In particular, this method led to an approach for inferences on orderings.

• As an addendum, a basic method for classifying documents using a distance-based approach (in Section 6.3) was proposed.

Although this thesis was originally motivated by the DEEDS data set, we found that our methods worked well for sequencing various sets of drafts although our sequencing results were also useful for DEEDS documents. Also, we saw that when the sequencing results from the Take and Put One algorithm were poor, most of the time the Markov Chain Monte Carlo method provides substantial improvement. Amongst the various distance measures that were described in Chapter 2, we found LSA-based distances between documents to be the most fruitful for sequencing problems. Ideally, we would have liked for the Tableaux-based distances to have been more effective for sequencing in our work, although they did prove useful for classification problems. The Tableaux method could possibly be improved by some reparametrization of the similarity measure (rather than just using the $\alpha$, $\beta$, $\gamma$ exponents in the formula for similarity) or by constructing a better technique to
choose the optimal parameter values that will perform well in sequencing (compared to the method proposed in Section 2.5.2). Finally, although the methods proposed for finding distances that align with time seemed promising, it was not always adequate for the particular data sets we worked with. We hope however that this method could be applied to other data sets more effectively and/or can lead us to devise new techniques for identifying distance measures that would be useful for sequencing purposes. Furthermore, it would be of value to create such a method that does not require the use of a training set. This is a challenging problem but necessary for cases that have an insufficient amount of documents with known dates. Our contributions, overall, should be placed in the context of other work on the problem of sequencing, as described in the literature reviewed in Chapter 1.

7.2 Open Problems and Future Work

Some open problems that were proposed but not fully solved in this thesis were mentioned in Section 2.4. In particular, a logistic SVD model was suggested in Section 2.4.2. Our future work involves searching for appropriate components of the document and term feature vectors that can be used to model incidence counts in a Document-Term matrix. We believe that this fitting procedure, although similar to SVD is more complicated, but can be fit feature by feature.

In some cases, it is feasible to measure different types of distances between documents. Each of these distance measures may give possibly different estimated orderings when used in an optimization algorithm, and some distances will naturally be more effective than others. It may be useful to construct an Ensemble type method that combines sequencing results based on different distances or even based on different sequencing algorithms. Another similar problem is testing the alignment of two given orderings. Specifically, suppose there are two different data sets for which of each is given an (estimated) ordering, and that one wishes to determine whether the two orderings are in the same direction (both sequenced in forward/backward chronological order or if one is ordered forward and the other backwards). If one could devise a technique to accomplish this, then sequencing of very large data sets could be simplified by first dividing the data sets into smaller
groups, then sequencing within these groups, and finally interlacing the estimated orderings (i.e. combining all orderings together to get an overall estimated ordering) using the Fix T Take and Put One algorithm. This would be a worthwhile problem to solve since it would be convenient for very large data sets such as DEEDS and for documents that are already categorized (such as by type, author, language, or origin). This is because it will reduce computational intensity and will also be more effective since documents that belong to the same category will naturally be more similar to one another so results should be improved by separating the groups rather than sequencing all the documents together. We mention that the problem of alignment potentially arises in application areas other than the ones considered here.

A method was proposed in Section 6.3 to classify documents when a training set is available. This method could be extended so that it could be applied without training data. This problem is related to Multidimensional Scaling in that it involves grouping the most similar elements together.

This thesis focused on distance-based methods for sequencing documents, but we seek to explore different methods in the future. As mentioned in Section 1.3, terms have different patterns of occurrence over time (some words/phrases become obsolete, occur very frequently during a certain period of time, or appear constantly throughout time such as non-contextual words). Thus, one alternative method for sequencing could involve examining the distribution of terms over time. For each term in the corpus, a graph of its distribution is made for each ordering. Intuitively, the ordering that has the smoothest distribution would be the optimal ordering. However, there are many different terms and graphs which complicates the process. This calls for a method to combine relevant information from each term to obtain an overall distribution for each ordering.

7.3 Potential Applications

The methods of this thesis can potentially be applied or extended for use in different sequencing problems not explored here. Some examples include following news stories, sequencing events on the internet, DNA sequencing/phylogenetic trees, etc. A phylogenetic tree is a tree that displays
the relationships between biological species based on their similarities and differences. Distance-based methods can be used to measure similarity between species in their physical and genetic characteristics in order to construct the tree.

The techniques that we have devised and studied in this thesis can also be used in various applications other than sequencing. There are many applications to text mining and information retrieval such as plagiarism detection, authorship detection, spam filtering, spell checking, and identity fraud on the internet. Different types of distances such as the ones we have proposed can be used to measure similarity between documents that would help determine whether two documents use almost all the same ideas, were written by the same author, or if an email should be sent to the spam folder. In some sense, the above all represent specialized versions of classification problems.

In this thesis we have attempted to make a contribution to distance-based methods for sequencing problems and have added some unique techniques that have not been entirely explored in the literature. Moreover, these techniques can be applied to other broader ranges of problems in text mining, information retrieval, and even such fields as biology and linguistics.
Appendices
Appendix A

An Example Using the Tableaux Method

To demonstrate the use of the Tableaux method for obtaining distances between documents, we present a simple example. Consider a corpus with $n = 3$ documents:

$D_1$: “It is raining today and tomorrow.”

$D_2$: “It is raining today but not raining tomorrow.”

$D_3$: “It is not raining but I brought my umbrella!”

Tableau for Document 1 (length $n_1 = 6$) and Document 2 (length $n_2 = 8$):

<table>
<thead>
<tr>
<th>Shingle Number $(s)$</th>
<th>Shingle Name</th>
<th>Shingle Length $(\ell_s)$</th>
<th>Count $(N_s)$</th>
<th>Proportion $(p_s)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>it is raining today</td>
<td>4</td>
<td>1</td>
<td>2/3</td>
</tr>
<tr>
<td>2</td>
<td>tomorrow</td>
<td>1</td>
<td>1</td>
<td>2/3</td>
</tr>
</tbody>
</table>
APPENDIX A: AN EXAMPLE USING THE TABLEAUX METHOD

Tableau for Document 1 (length $n_1 = 6$) and Document 3 (length $n_3 = 9$):

<table>
<thead>
<tr>
<th>Shingle Number $(s)$</th>
<th>Shingle Name</th>
<th>Shingle Length $(\ell_s)$</th>
<th>Count $(N_s)$</th>
<th>Proportion $(p_s)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>it is</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>raining</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Tableau for Document 2 (length $n_2 = 8$) and Document 3 (length $n_3 = 9$):

<table>
<thead>
<tr>
<th>Shingle Number $(s)$</th>
<th>Shingle Name</th>
<th>Shingle Length $(\ell_s)$</th>
<th>Count $(N_s)$</th>
<th>Proportion $(p_s)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>it is</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>not raining</td>
<td>2</td>
<td>1</td>
<td>2/3</td>
</tr>
<tr>
<td>3</td>
<td>but</td>
<td>1</td>
<td>1</td>
<td>2/3</td>
</tr>
</tbody>
</table>

At a first glance, we can see that documents 1 and 3 should be the least similar as they only have 2 relatively short shingles in common and both shingles occur frequently in the corpus. Documents 1 and 2 share the longest shingle (of length 4) when compared to other pairs of documents in the corpus, whereas documents 2 and 3 have the most common shingles. It is difficult to say which pair (1,2) or (2,3) are more similar as this may depend on the choice of parameters $\alpha, \beta, \gamma$ from Equation 2.5.3. Documents 1 and 2 will have a stronger similarity because of the shingle length $(\ell_s)$ and so the bigger the $\alpha$, the more similar they will be. Documents 2 and 3 will have a high similarity because they have the most number of shingles in common (higher value for $m_{ij}$ so more terms in the sum will contribute to the similarity measure). Note that contextual words such as “umbrella” and “brought” are not listed in any of the tableaux as they appear in only one document and therefore are not considered important when measuring similarity in the entire corpus.

Using the parameter values $(\alpha, \beta, \gamma) = (2, 1, 0.5)$ and substituting into Formula (2.5.3), we obtain the following un-normalized similarities between document pairs: $\text{sim}(1, 2) \doteq 3.680608$, $\text{sim}(1, 3) \doteq 0.6804138$, $\text{sim}(2, 3) \doteq 1.355288$. We can see that documents 1 and 2 are the most similar when compared to other pairs in the corpus, and then documents 2 and 3 are most similar, and finally
documents 1 and 3 are quite dissimilar, as expected. After normalizing and converting to distance measures using the method defined in (2) from Section 2.6, we get these pairwise distances: $d_{1,2} = 0$, $d_{1,3} = 1.414214$, $d_{2,3} = 1.245035$. Note that documents 1 and 2 have a distance measure of 0 even though they are not identical. This is because the distance measures were calculated by normalizing the similarities and so the distances are relative to the entire corpus. Here, the distance of 0 signifies that documents 1 and 2 are the closest in the entire corpus and does not necessarily mean the documents are identical. The resulting measure is a dissimilarity measure. This should not be a problem since our sequencing methods do not rely on the pairwise distances being metrics (refer to Section 4.2).
Appendix B

DEEDS Charter Format and Computer Programs

Each charter in the entire DEEDS corpus has a unique identification number, date (the year if already dated or ‘0000’ if undated), type, and possibly other information such as whether the date was internal or assigned, etc. Each document that was available for use in this thesis was formatted as follows:

# "Document ID", "Date", "Type"
/
document text
/

Each document header (which contains the characteristic information for a charter) begins with a ‘#’ sign and the contents of each document is between two ‘/’s for easy identification. Below is an example of how a DEEDS document looks:
All programming and simulations were done using the ‘R’ Statistical Software, Version 3.0.1. The raw data was made available to us as one file with all document contents formatted as above. An ‘R’ program was coded to first dismantle the file and make a separate file for each of the documents with their respective contents. In some cases, each file was put into the same large corpus, whereas in other cases, a separate corpus was created for each type (depending on the analysis performed). The ‘tm’ package (Feinerer, 2012) was used for basic text mining functions such as reading the corpora and performing LSA, and the package ‘nnls’ (Mullen and van Stokkum, 2007) was implemented for
Non-Negative Least Squares optimization in the Take and Put One algorithm. All other programs including the Tableaux method, Take and Put One algorithms, and MCMC procedures were coded without the use of built-in functions.
Bibliography


