A thesis submitted in partial fulfilment of
the requirement for the degree of Doctor of Philosophy

On the Helmholtz Principle for Text Mining

Boris Dadachev

School of Mathematics, Cardiff University
& Hewlett-Packard Laboratories, Bristol

October 2015

Declaration

This work has not been submitted in substance for any other degree or award at this or any other university or place of learning, nor is being submitted concurrently in candidature for any degree or other award.

Signed .... (candidate)

Date ....

Statement 1

This thesis is being submitted in partial fulfillment of the requirements for the degree of PhD.

Signed .... (candidate)

Date ....

Statement 2

This thesis is the result of my own independent work/investigation, except where otherwise stated. Other sources are acknowledged by explicit references. The views expressed are my own.

Signed .... (candidate)

Date ....

Statement 3

I hereby give consent for my thesis, if accepted, to be available online in the University's Open Access repository and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed .... (candidate)

Date ....

To Natalia.

Abstract

The majority of text mining systems rely on bag-of-words approaches, representing textual documents as multi-sets of their constituent words. Using term weighting mechanisms, this simple representation allows to derive features that can be used as input by many different algorithms and for a variety of applications, including document classification, information retrieval, sentiment analysis, etc. Since the performance of many mining algorithms directly depend on term weights, techniques for quantifying term importance are of paramount importance in text processing.

This thesis takes advantage of recent advances in keyword extraction mechanisms, which further select the terms with the highest weights to keep only the most important words. More precisely, building on a recent keyword extraction technique, we develop novel text mining algorithms for information retrieval, text segmentation and summarization. We find these algorithms to provide state-of-the-art performance using standard evaluation techniques. However, contrary to many state-of-the-art algorithms, we try to make as few assumptions as possible on the data to analyze while keeping good computational performances, both in terms of speed and accuracy. As such, our algorithms can work with inputs from a variety of domains and languages, but they can also run in environments with limited resources. Additionally, in a field that tends to be dominated by empirical approaches, we strive to rely on sound and rigorous mathematical principles.

Supervisors:

- Prof. Alexander Balinsky, Cardiff School of Mathematics, Cardiff (UK).
- Dr. Helen Balinsky, Hewlett-Packard Laboratories, Bristol (UK).
- Dr. Steven Simske, Hewlett-Packard Laboratories, Fort Collins (USA).

Acknowledgements

I owe my gratitude to my supervisors, Prof. Alexander Balinsky, Dr. Helen Balinsky and Dr. Steven Simske, for their guidance, expert advice and availability over the past few years. While keeping me on the right track, they have also helped me learn something new every day and to develop myself as a researcher.

I am also grateful towards my peers and the staff from Cardiff University and HP Labs, for making these two places such pleasant and friendly environments.

I would like to acknowledge and thank all my sponsors as well. Hewlett-Packard Laboratories and the Cardiff University School of Mathematics have funded this research project, allowing me to complete this PhD. Furthermore, ACM SIGWEB has helped me fund my travel to the DocEng 2014 conference.

Finally, I wish to extend my sincerest thanks to my family for their incommensurable help and support through the years.

Dissemination of Results

Peer-Reviewed Publications

- B. Dadachev, A. Balinsky, H. Balinsky, and S. Simske. On the Helmholtz Principle for Data Mining. In *Proceedings of the 3rd IEEE Conference on Emerging Security Technologies*, 2012.
- B. Dadachev, A. Balinsky, H. Balinsky, and G. Forman. Automatic Text and Data Stream Segmentation using Weighted Feature Extraction. In *Proceedings of the 3rd IMA Conference on Mathematics in Defence*, 2013.
- H. Balinsky and B. Dadachev. Automatic Text and Data Stream Segmentation using Principles of Human Perception. In *Proceedings of the HP Imaging and Color Symposium*, 2013.
- B. Dadachev, A. Balinsky, and H. Balinsky. On Automatic Text Segmentation. In *Proceedings of the 14th ACM Symposium on Document Engineering*, 2014.

Patents

- H. Balinsky, A. Balinsky, B. Dadachev, and S. Albright. Keywords to Generate Policy Conditions. US Patent Application 2014/44596, filed June 27, 2014. Patent pending.
- H. Balinsky, A. Balinsky, B. Dadachev, and S. Simske. Document Access. US Patent Application 2015/67807, filed August 3, 2015. Patent pending.
- H. Balinsky, B. Dadachev, S. Simske, and A. Balinsky. Identifying Documents. US Patent 2015/46324, filed August 21, 2015. Patent pending.

Talks and Presentations

- Helmholtz Principle and Small-World Networks for Summarization. *Wales Mathematics Colloquium.* Gregynog (UK), May 23, 2012.
- On the Helmholtz Principle for Data Mining. *IEEE Conference on Emerging Security Technologies.* Lisbon (Portugal), September 6, 2012.
- Automatic Document Segmentation. *Wales Mathematics Colloquium.* Gregynog (UK), May 21, 2013.
- Automatic Text and Data Stream Segmentation using Weighted Feature Extraction (Poster). *IMA Conference on Mathematics in Defence.* Malvern (UK), October 24, 2013.
- Automatic Text and Data Stream Segmentation (Poster). *Set for Britain event for Early-Stage Researchers.* Parliament of the United-Kingdom, March 17, 2014.
- Mining Text with the Helmholtz Principle and Example Applications. *Computer Science seminar, Royal Holloway University of London.* Egham (UK), June 3, 2014.
- On Automatic Text Segmentation. *ACM Symposium on Document Engineering.* Fort Collins (Colorado, USA), September 18, 2014.

Others

- H. Balinsky and B. Dadachev. Automatic Text Segmentation (Poster). *HP Labs Poster and Demo Fair.* Palo Alto (USA), April 2014.
- H. Balinsky and B. Dadachev. Enterprise Search as a Light Cloud Service (Poster). *HP Labs Poster and Demo Fair.* Palo Alto (USA), April 2014.
- H. Balinsky, D. Perez, and B. Dadachev. Comprehensive Solution for Handling Sensitive Data. *HP Labs Poster and Demo Fair.* Palo Alto (USA), June 2015.
- H. Balinsky and B. Dadachev. Invenies: Enterprise Search. *HP Labs Poster and Demo Fair.* Palo Alto (USA), June 2015.

Contents

1 Introduction **1**

2 Background: Text Mining **3**

2.1 Preprocessing 4

2.1.1 Pre-requisites 4

2.1.2 Preprocessing tasks and techniques 4

2.1.3 Discussion 7

2.1.4 Our approach to preprocessing 8

2.2 Text mining algorithms and applications 8

2.2.1 Numerical representation of documents 9

2.2.2 Common mining algorithms and applications 10

2.2.3 Current research challenges and discussion 12

2.2.4 Our approach to text mining 13

3 Background: the Helmholtz principle **14**

3.1 Gestalt theory and Helmholtz principle for computer vision 14

3.1.1 Visual perception and Gestalt theory 14

3.1.2 The Helmholtz principle for computer vision 16

3.2 Helmholtz principle for data and text mining 18

3.2.1 NFA for m-tuples in a balls-and-bins experiment 18

3.2.2 Practical adaptation of the balls-and-bins model to keywords extraction 19

4 Information Retrieval and Enterprise Search 23

4.1 A brief literature review 24

4.1.1 Indexing 24

4.1.2 Retrieval models 25

4.1.3 Learning-to-Rank 28

4.2 Proposed algorithm 29

4.2.1 Preprocessing 29

4.2.2 Term weighting scheme 30

4.2.3 Augmenting the system with Learning-to-Rank 31

4.3 Evaluation methodology 36

4.3.1 Evaluation procedure 36

4.3.2 Evaluation metrics 37

4.4 Experimental results on the TREC datasets 39

4.4.1 The CERC collection and associated TREC datasets 39

4.4.2 Initial ranking evaluation 40

4.4.3 Re-ranking top results with learning-to-rank 44

4.5 Conclusion 49

5 Text Segmentation 50

5.1 A brief literature review 51

5.2 Proposed algorithm 55

5.2.1 Preprocessing 55

5.2.2 Gap scores calculation 55

5.2.3 Identification of segment boundaries 58

5.3 Evaluation methodology 61

5.3.1 Evaluation procedure . . . 61

5.3.2 Evaluation metrics . . . 62

5.4 Experimental results . . . 63

5.4.1 Synthetic datasets . . . 63

5.4.2 Towards more realistic datasets . . . 66

5.5 Conclusion . . . 75

6 Automatic Summarization 76

6.1 A brief literature review . . . 76

6.2 Proposed algorithm . . . 80

6.2.1 Preprocessing . . . 80

6.2.2 Keyword extraction . . . 80

6.2.3 Sentence extraction . . . 81

6.2.4 Complete description of the original algorithm . . . 83

6.2.5 Extending the original algorithm using affiliation networks . . 84

6.2.6 Centrality measures . . . 88

6.2.7 Complete algorithm description . . . 90

6.3 Evaluation methodology . . . 91

6.3.1 Evaluation procedure . . . 91

6.3.2 Evaluation metrics . . . 92

6.4 Experimental results . . . 94

6.4.1 Comparative evaluation on the DUC 2002 dataset . . . 94

6.4.2 Experiments with longer documents . . . 99

6.5 Conclusion . . . 103

7 Conclusion 104

Appendices 106

A Demos and pilot projects 106

A.1 Enterprise Search 106

A.2 Segmentation 110

A.3 Summarization 113

Chapter 1

Introduction

As the cost of computer storage keeps decreasing and our modern societies get increasingly connected, the amount of textual data available on the internet and the companies' intranets is growing faster than ever. However, this proliferation of data comes with a number of challenges, from both the data exploitation and data security points of view. These numerous challenges have kept mathematics and computer science researchers busy for decades, as the vast field of text mining was developing. And if everyone agrees that these data are potentially great sources of information and intelligence, processing them remains difficult to this day. The main challenges are related to the very diverse nature of these data (in terms of languages, jargons, types of documents, etc.) as well as, often, their sheer volume. To make things worse, the large number of text mining applications, ranging from consumer and business intelligence to information security, makes text mining systems complex and reliant on a number of different algorithms.

However, the first step in text processing is almost always to transform the input text into a numerical representation that text mining algorithms can work with. To this end, bag-of-words approaches that model a piece of text as a multiset of words have become nearly ubiquitous, even if they are not suitable for all problems. Ignoring the order in which words appear may seem crude but in practice, bag-of-words approaches have proven to be extremely robust and efficient for most problems. Then, relative word importance is usually determined using a term weighting scheme like the well-known TF-IDF and term weights are used as input for text mining algorithms. Therefore, the performance of most text mining algorithms directly depends on the quality of the term weights they rely on internally.

A threshold can also be applied to these term weights in order to keep the most important words only or, said differently, to extract keywords. Recently, Balinsky et al. proposed a novel approach to keyword extraction [21, 22] based on ideas from computer vision and specifically on the Helmholtz principle from the Gestalt theory of visual perception [47]. The Helmholtz principle provides a sound and computationally efficient statistical framework to identify meaningful words using non-parametric methods. The resulting keywords and their associated "level of meaningfulness" can then be used as input by text mining algorithms. These were shown to be effective [22] and the authors proposed an application to summarization, which relies on the extracted keywords to model documents as small-world graphs [24, 25].

The main goal of this thesis is to build on this keyword extraction mechanism. We investigate several potential applications and see how this keyword extraction technique can be used to improve current text mining algorithms and systems. We propose a number of new algorithms and thoroughly verify their effectiveness using standard evaluation techniques. We strive to develop algorithms that are:

- *Generic*. Making as few assumptions on the type of data to analyze as possible allows to design algorithms that are not limited to restricted inputs in terms of quality, language, jargon, etc.
- *Fast and scalable*. Many advanced algorithms have been proposed over the years. While these may be particularly effective they may also be limited to relatively small inputs because they are computationally expensive and/or because they do not scale to larger inputs.
- *Theoretically sound*. Text mining is a field that is often dominated by empirical approaches rather than rooted on sound mathematical principles.

More precisely, this thesis is outlined as follows. Chapter 2 gives a brief overview of the field of text mining and introduces preprocessing techniques and common mining algorithms. Chapter 3 familiarizes the reader with the Helmholtz principle, explained in its original context as well as its recent adaptations to computer vision and text mining. Subsequent chapters each present new algorithms and applications. Chapter 4 proposes a new term weighting scheme for information retrieval purposes and describes a full search engine for enterprise search built around it. Chapter 5 continues with a new algorithm for text segmentation, which effectiveness is demonstrated for a variety of inputs and languages. Chapter 6 extends the summarization algorithm introduced in [24]; new models are developed and a comprehensive evaluation is performed. Chapter 7 concludes this work.

Finally, as this is an industrial PhD, the work has also lead to a number of proof-of-concepts and pilot programs, some of which are described in Appendix A.

Chapter 2

Background: Text Mining

Text mining, also referred as text analytics in business-oriented settings, is a broad and relatively recent research area which aims at *automatically extracting and structuring the information* contained in (usually large) collections of text. In the current Information Age, large amounts of textual data are routinely collected and stored. Those data are potentially rich sources of information but their sheer amount also creates an information overload. This is the reason why text mining is finding its way in nearly all businesses and industries in one form or another. Indeed, text mining systems allow to process, search and visualize information, monitor trends, detect sentiment and anomalous events, etc. As such, text mining is at the cross-roads between many disciplines, such as data mining, machine learning, statistics, information retrieval, computational linguistics and natural language processing.

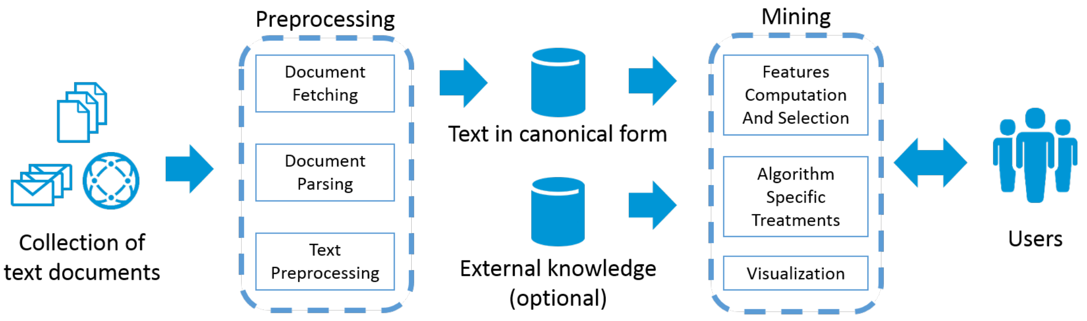

Preprocessing
Mining
Document Fetching
Document Parsing
Text Preprocessing
Text in canonical form
External knowledge (optional)
Features Computation And Selection
Algorithm Specific Treatments
Visualization
Collection of text documents
Users

Figure 2.1: Simplified, high-level architecture of a text mining system

As shown in Figure 2.1, text mining systems have two main steps. The first one, referred as preprocessing, deals with extracting the text from a collection of documents and converting it into a well-defined canonical format for further processing. The second step is the mining step per se. The next sections describe these two steps in more details.

2.1 Preprocessing

Preprocessing is a preliminary step that is of paramount importance for subsequent mining algorithms to give good results. Let us briefly review the most common preprocessing techniques.

2.1.1 Pre-requisites

The first two tasks in any text mining system consist of:

1. fetching the documents from their original location, and
2. extracting the text from each document (i.e., parsing the original documents).

A large number of problems can occur during these stages. Common issues arising when fetching documents include network errors and large latencies for distant document repositories. Potential parsing problems are also numerous, due to unknown file encodings or formats, encrypted files or even scanned documents where the text is in the form of an image. Information security can also be a major concern when dealing with confidential or sensitive material. In some cases, all Personal Identity Information (PII) or confidential material must be removed prior to any processing in order to prevent potential information leaks. These initial steps may therefore require important development efforts in practice. However, they are system specific and will not be further mentioned in this thesis.

2.1.2 Preprocessing tasks and techniques

At this stage, the running text (i.e., the document body) has been extracted from the original documents and is in the form of a sequence of characters. The goal of preprocessing is to convert those raw sequences of characters into a canonical form, ready for subsequent mining algorithms. Given the broad goals of text mining and the broad nature of the algorithms involved, different systems will have different preprocessing requirements. The most common preprocessing techniques include tokenization, word normalization, stop-words removal and stemming.

It is important to note that all preprocessing techniques are language specific. The work in this thesis mainly focuses on mining English documents, and so is the remaining of this section.

Tokenization

Before any mining algorithm can be applied, the running text in each document —in the form of raw strings— must be broken into smaller meaningful units called *tokens*. This process is called *tokenization*. Words are "the smallest meaningful units that can stand on their own" (see page 11 in [67]) so very naturally the vast majority of systems will require tokenization at the word level. However, some systems may have different requirements. In addition to or in replacement of word tokenization, it is also common to divide the running text into sentences, clauses or even overlapping character n-grams.

Tokenization is a complex task that depends on the exact definition given to tokens, especially as difficult cases and exceptions are numerous. For example, "I'm" should be segmented into "I" and "m" (or "am") but should the name "O'Connell" also be split because of the apostrophe? What about proper nouns (e.g., "Palo Alto"), compound words (e.g., "part-of-speech") and abbreviations (e.g., "q.e.d.")? The simplest tokenization schemes use a predefined set of characters as token delimiters. For word tokenization, such delimiters may be white space or non-alphabetical characters. More accurate approaches can be rule-based [13, 121] or use machine learning techniques [68], at the expense of some additional work.

Let us illustrate word tokenization with an extract from "Alice in Wonderland" [7]. The following two sentences:

You used to be much more... muchier. You've lost your muchness.

are transformed into the following list of tokens:

"You", "used", "to", "be", "much", "more", "muchier", "You", "ve", "lost", "your", "muchness".

Word normalization

After word tokenization, the same word can be found in the list of tokens with slight orthographic and case differences. *Normalization* tries to remove these variations. American English and British English spellings are a common example: clearly, "airplane" and "aeroplane" are the same word. But what about "apartment" and "flat"? Words of foreign origin can also sometimes be found in various spellings, such as "cafe" and "café".

In most cases, normalization is limited to case folding (e.g., lower-casing all words) but more advanced algorithms also exist. The reader may refer to the ones proposed by Sproat et al. [122] and Clark and Araki [37] for concrete examples.

For the previous example, limiting normalization to lower-casing lists the following tokens:

"you", "used", "to", "be", "much", "more", "muchier", "you", "ve", "lost", "your", "muchness".

Note that a more complex normalization procedure could have replaced "ve" by "have".

Stop-words removal

A number of common words called *stop-words*, such as determiners and prepositions (e.g., "a", "the", "of", "with"), carry very little meaning in a text. Since such words are also very frequent and they tend to appear in any text, most systems remove them during preprocessing in order to speed up computations and/or because some mining algorithms perform poorly in their presence. However, stop-words filtering has a number of negative consequences and is certainly not adequate for all applications, as will be discussed in subsequent chapters. Also, there is no canonical list of stop-words: lists commonly used in English range in size from a few dozen to several hundred words (see Section 2.2.2 in [86]).

Continuing with our running example, stop-words removal leaves:

"used", "much", "muchier", "ve", "lost", "muchness".

It is interesting to note that half the words have been discarded with the stop-words list used to generate this example. Also, "ve" was not removed but "have" would have been removed.

Stemming

It is extremely common for text mining systems to reduce inflections[1] and some derivations[2] to their roots, in a process called *stemming*. The most popular stemming algorithm for the English language is Porter stemmer [107]. The interested reader may refer to Chapters 4 and 5 in [34] for a linguistically rigorous introduction to stemming.

[1]New words formed from grammatical variations, e.g., due to tense and and number.
[2]New words formed by adding prefixes and suffixes.

The main reason for stemming is to reduce the vocabulary size by mapping "similar" tokens together and thus to consider closely related words as the same (e.g., "dog" and "dogs"). Note that stemming is often a rather approximative process that strips suffixes according to a predefined set of rules. On the one hand, this makes stemming very fast. On the other hand, the resulting stems may not always be linguistically correct due to numerous language irregularities. More sophisticated algorithms called lemmatizers (see [140] for an example) take into account the context in order to reduce words to their lemma (i.e., dictionary form). However, when compared to stemmers, lemmatizers often bring little benefits to the overall performance of mining systems. This is the case, for example, in information retrieval (see Section 2.2.4 in [86]).

On our running example, the Porter stemmer gives:

"use", "much", "muchier", "ve", "lost", "much".

Note that "used" becomes "use" but the irregular verb "lost" is left as is.

Other techniques

There are other relatively common preprocessing techniques that will not be used in this work. Let us mention them briefly for completeness. Part-of-speech tagging consists in annotating each word with its part-of-speech [38]. Parsing goes one step further and constructs the entire syntactic tree for each sentence [69] in order to identify the different sentence components and their relations; it mainly helps for disambiguation. Finally, entity recognition and extraction [88] can also be used for more specific analyses. For example, one may need to identify and extract all company names, locations and dates in a text. Entity extraction is often an important part of practical systems as they add shallow semantic information. However, they are very domain specific and thus system specific, as they rely on hand-written rules or, more often nowadays, supervised learning techniques.

2.1.3 Discussion

At a more abstract level, preprocessing serves two concurrent objectives. On the one hand, it reduces in a controlled way the vocabulary size and thus the number of features that subsequent mining algorithms use. By its very nature, text is extremely sparse and reducing the dimension of the feature space helps with a number of theoretical and practical problems such as the curse of dimensionality (see page 97 in [29]). On the other hand, preprocessing is a destructive procedure. Indeed, since in most cases the original text cannot be recovered after preprocessing, some information has been lost. Given the incredible complexity of language, its ambiguities and subtleties, destroying information can be potentially damaging for the end results. The right balance will depend, once again, on the mining algorithms used and applications of the system.

2.1.4 Our approach to preprocessing

Regarding preprocessing, several examples where the sophistication of the algorithm has very little influence on the end results of a text mining system have been mentioned. For that reason and since our goal is to build robust generic algorithms, we choose a very pragmatic approach in our work, relying only on simple and fast techniques that were proven to work well in most settings:

- Word tokenization is performed by splitting the running text on every non-alphabetical characters. Sentence tokenization, when necessary, uses all occurrences of the characters full stop ("."), exclamation mark ("!") and question mark ("?") as sentence delimiters.
- The only normalization used is case-folding: all tokens are converted to lower case.
- In most cases, no stop-words are removed. Cases where stop-words filtering is necessary will be clearly mentioned.
- Finally, we use Porter stemmer [107] for stemming purposes.

2.2 Text mining algorithms and applications

Let us present some of the most common applications of text mining and the associated underlying algorithms. Note that text mining is a very large field of research and therefore also a relatively generic term. As such, it would not be possible to comprehensively review all applications and algorithms. This section is intended as a light introduction for notions and techniques that will be mentioned throughout this work; more specialized reviews are also provided in later chapters as different fields of application are investigated.

2.2.1 Numerical representation of documents

After preprocessing, one needs to transform the cleaned input into a numerical representation that subsequent mining algorithms can work with. The simplest and most prevalent model is the so-called *bag-of-words*, which represents a piece of text as the multiset of its constituent words. As an illustration, the bag-of-words for our running example from the previous section is:

$$\{\text{"lost": 1, "much": 2, "muchier: 1, "use": 1, "ve": 1}\}.$$

Thus this model ignores the order in which words appear as well as grammatical information. Those simplifying assumptions then allow to represent a piece of text (e.g., a document) as a vector where each coefficient corresponds to a word and quantifies the importance of this word. A large variety of *weighting schemes* can be used for the coefficients. Among the most common are:

- binary weights, indicating whether the word appears in the document,
- frequency weights (possibly normalized) indicating the number of times the word appears in the document, or
- more complex weights like TF-IDF [118].

Such simple vector representations of text may seem naive given the complexity of natural language but they have proven extremely successful for many applications, like information retrieval, document classification, summarization, and many more. An in-depth review of weighting techniques for information retrieval is done in Chapter 4.

Yet, bag-of-words models have several drawbacks. First, they miss semantic relations between words; indeed, synonyms, antonyms and other related words are treated like completely independent terms. Furthermore, the resulting vectors are very sparse as most documents contain a few dozens or hundreds of unique words, whereas vocabulary sizes are typically in the dozens of thousands of words. *Dimensionality reduction* techniques help to solve both issues: capturing semantic links between words while compressing the data at the same time.

Latent Semantic Analysis (LSA) [46] is one of the most popular technique to do so. Given a collection of documents, one can compute a term-document matrix A where A_{ij} is the weight for the word i in document j (for example a TF-IDF weight). Then, Singular Value Decomposition (SVD) can be used to compute a low-rank approximation of the matrix A. Intuitively, this procedure projects the original data in a smaller space where each dimension is a semantic "concept". These concepts are linear combinations of the original terms, so they can be difficult to interpret. LSA has a wide range of applications, although it was originally proposed for information retrieval where it allows to build smaller indexes and also to increase search results recall by mapping related terms together. However, LSA is computationally expensive, especially for dynamic collections of documents where the term-document matrix regularly changes. Topic modeling is another popular dimensionality reduction technique. Here, each document is seen as a mixture of topics, where topics are clusters of words that often co-occur in different documents. Given a number of topics and a collection of documents, topic modeling automatically identifies the topics (i.e., co-occurring words) and does a soft clustering assignment for each document. The most popular algorithms are Probabilistic Latent Semantic Analysis (PLSA) [62] and Latent Dirichlet Allocation (LDA) [31].

Dimensionality reduction solves some of the bag-of-words problems but not all. For example, ignoring word orders is simply inappropriate for some applications. Natural language generation is a clear example: "the brown dog" is not equivalent to "dog the brown". For such applications, it is common to work with n-grams of words (e.g., bi-grams or tri-grams, that is, two- or three-tuples of consecutive words) [3, 91].

2.2.2 Common mining algorithms and applications

Let us briefly present some of the most common applications of text mining together with the associated popular approaches.

Searching large collections of text for a piece of information is a natural and very common requirement for text mining systems. With regard to search, text mining is very different from classical data mining. Indeed, textual data is considered *unstructured* due to its lack of well-defined structure, making relational databases not readily appropriate for storing and querying text. Instead, making text searchable and easily manageable is a challenge and the goal of a discipline called *information retrieval*. Vector representations of text make retrieval easy: as both documents and queries are vectors that lie in the same vector space, the documents that match closely a query will have a large cosine similarity. This model is called the vector space model, and explained in more details in Chapter 4.

Another omnipresent need in text mining is *categorization*. Categorization algorithms automatically identify the class, or classes, of an observation (e.g., a document) from a pre-defined set of possible classes. To name only a few examples, classification can be used for spam detection, automatic email categorization, genre, topic, sentiment and language detection, as well as automatic generation of meta-data. Historically, categorization was performed using sets of rules manually written by experts but those tend to be difficult and expensive to maintain. This is why supervised learning approaches are often preferred; a large number of algorithms are nowadays available, such as decision trees, linear classifiers and artificial neural networks [30]. Supervised learning methods require labeled data, usually in large amounts for optimal performance [57]. However, such labels are often both time-consuming and expensive to obtain. In such cases, semi-supervised learning methods take advantage of large amounts of unlabeled data to improve the classification performance obtained with a small set of labeled data [124]. Also, when only little labeled data is available for the domain of interest, transfer learning (and sometimes machine translation) can leverage resources from better datasets, though these techniques remain in their infancy [103].

There are many other common text mining tasks and widely used classes of techniques:

- With unlabeled data, unsupervised learning techniques, can be used to find patterns in the data [123]. In particular, *clustering* seeks to group similar observations together. For example, one could group documents that contain similar terms (and maybe discuss the same topic) for exploratory analysis or to organize a collection of documents. Also, note that dimensionality reduction techniques that were already mentioned can be seen as soft clustering.
- *Summarization* is another omnipresent need that seeks to condense long documents into shorter versions while preserving important information. Chapter 6 is dedicated to summarization and an in-depth review is presented on this occasion.
- *Information filtering* tries to remove uninteresting information from information streams. Showing news articles that only matches users' interests is an example [105]. More recently, these techniques have inspired research on recommender systems that most often work by finding items that similar users like.
- *Time series* analysis and *anomaly detection* methods are also widely used to analyze trends, for example on social media [87], or to detect potential threats in cyber-security and bio-surveillance contexts [39].

2.2.3 Current research challenges and discussion

Despite the large attention it has received from researchers, text mining still faces many challenges. Researchers generally agree that major improvements will come from better and deeper semantic understanding. So far, *semantic analysis* remains relatively superficial as it mainly stems from a better grammatical analysis of sentences, like parsing and co-reference resolution. The addition of external knowledge, for example from ontologies, can also help; well-known knowledge sources include WordNet [94] and the MeSH controlled vocabulary [116].

Real semantic analysis will require a much more advanced knowledge of the world, on what is possible and what not, what is likely and what not, etc. However, how to learn this advanced knowledge remains an open question, which is closely related to advances in artificial intelligence. The current trend in moving towards deeper semantics is building large semantic networks from the internet [60].

Different text collections are naturally very disparate and dealing with this disparity is another challenge. Indeed, different languages, contexts, jargons, specialized vocabularies, abbreviations, etc. are all complicating the text mining process. For example, scientists from different fields will refer to the same concept with different words. Also, social media are full of typos and non-standard abbreviations, making the text very noisy. Any algorithm trained on a particular domain will not generalize well to other domains. Transfer learning, which is still an active research area, is likely to play an important role in helping to dampen this issue [103].

Dealing with large quantities of data is yet another challenge. Many aspects become complicated as datasets grow, even the basics like storing and accessing the data efficiently. The efforts are currently focused on developing specialized software and on scaling algorithms using parallel and distributed computing frameworks, such as Map-Reduce [45] that can run on relatively inexpensive clusters of computers. Research is even being conducted to change computer architectures to deal more efficiently with data [12].

In summary, significant advances in text analysis are likely to come from the combination of advances in the various fields on which text mining relies.

2.2.4 Our approach to text mining

The remainder of this thesis will be focused on keyword-based algorithms, that is to say methods to identify important words in documents and compute a numerical measure of importance for each keyword. Bag-of-words methods might seem naive but as we have seen, they still power the vast majority of text mining systems and have several advantages over more complex approaches due to their simplicity, computational effectiveness and excellent practical results. Therefore, such approaches remain a major research area as they can have a direct impact on many mining systems.

This thesis is devoted to the development of algorithms that are:

- *Generic*. We have seen that text collections differ in their language, domain, quality, etc. Algorithms that require custom training data or a lot of manual tuning are expensive as they require the work of human experts. Generic algorithms that make few assumptions on the data and are therefore more robust provide compelling alternatives to manually tuned algorithms. Note that such generic algorithms can often be subsequently specialized using machine learning if needed; several practical examples will be provided in the coming chapters.
- *Fast and scalable*. This is important for two main reasons: so that the resulting algorithms can deal with large amounts of data, but also so that they require limited computational resources. Indeed, many devices such as printers and other embedded systems have very limited computing power that de facto prevents the use of many complex algorithms.
- *Theoretically sound*. The most common term weighting scheme is, by far, TF-IDF. Despite its good performance for a variety of tasks, TF-IDF remains rooted on empirical observations. We will strive to develop algorithms that have solid mathematical foundations.

Chapter 3

Background: the Helmholtz principle

Our work builds on a keywords extraction procedure introduced by Balinsky et al. [22]. This procedure's roots come from the Gestalt theory of visual perception. Let us present it briefly.

3.1 Gestalt theory and Helmholtz principle for computer vision

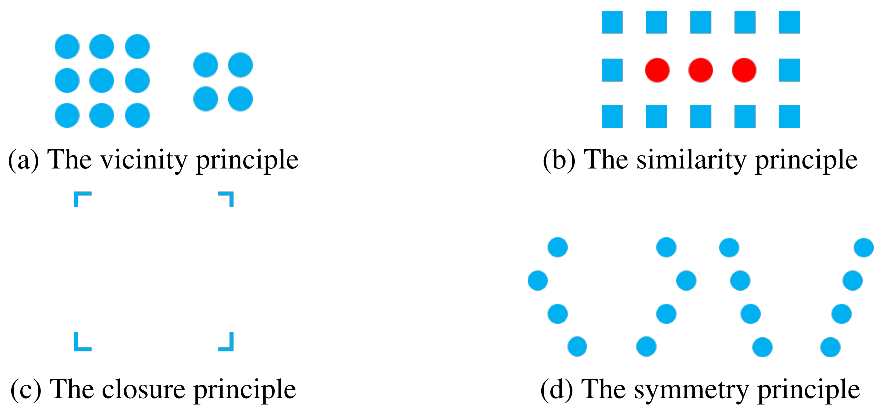
3.1.1 Visual perception and Gestalt theory

Visual perception is the science studying how the visual system processes and interprets the information contained in the light reaching the eyes. In particular, the retina is made of millions of photo-receptor cells that are sensitive to light and trigger electric reactions sent to the brain via the optic nerve (see page 25 in [33]). But how does the brain recover and interpret a full image from these millions of stimuli? The complexity of the task is huge: after all, every image we form in our brain is unique. Most objects in those images will be obviously familiar but they are seen under unique conditions (different angles of view, lighting, etc.). Yet we are able to make sense of what we see effortlessly. This fact has lead scientists to hypothesize that there must be general principles of perception, even though the exact mechanisms at play in image reconstruction remain, to date, mostly a mystery.

One of the most influential theories of visual reconstruction is the Gestalt theory [89, 134], whose development started in Germany in the 1920s. The central idea behind the Gestalt theory is that "the whole is other than the sum of the parts" [90]. Indeed, one often sees a line before seeing the collection of points that constitutes it. For Gestalt theoreticians, the fact that the entirety of an object is seen before its individual components is due to innate *grouping principles* [134]. These principles identify the way our brain automatically groups points —or retina stimuli— that share similar characteristics together to form a coherent whole, or *gestalt*. Recursively, similar gestalts will be grouped together into new and bigger gestalts.

The work of "gestaltists" has mainly consisted in creating many illustrations of the grouping principles and categorizing them [33]. Thus, there is not a universally accepted set of principles. For illustration purposes, let us present the most common ones. The interested reader may refer to Chapter 6, in [33], for a more comprehensive introduction.

- *The vicinity principle:* objects appearing close to each other are grouped together. In Figure 3.1a, points at a small distance from each other appear like a group, while increasing the distance between points creates distinct objects.
- *The similarity principle:* similar objects, in shape, color, texture, etc., are grouped together. The effect is illustrated in Figure 3.1b, where objects with similar shape and color are grouped together and separated from dissimilar objects.
- *The closure principle:* objects tend to be seen as a whole even when they are incomplete. Only the angles of a square are present in Figure 3.1c, yet the square is clearly visible.
- *The symmetry principle:* sets of objects that are symmetric tend to be seen as a whole. Four main groups of points are drawn in Figure 3.1d and two bigger objects emerge due to symmetry.


(a) The vicinity principle

(b) The similarity principle

(c) The closure principle

(d) The symmetry principle

Figure 3.1: Illustrations of the main grouping principles from the Gestalt theory. Adapted from [33].

The Gestalt theory has been criticized for being descriptive rather than explanatory and it has been quickly superseded in the field of psychology by newer theories such as Tolman's cognitive theory of learning [128]. However, the Gestalt theory remained popular as it provides compelling explanations for many visual effects and it still has an undeniable influence nowadays in the arts. It has also influenced some recent developments in computer vision.

3.1.2 The Helmholtz principle for computer vision

There is a striking similarity between visual perception and digital images. Indeed, eyes and digital cameras are both made of photo-receptors. Our brain reconstructs the images we see from a large number of retina cells stimuli and these stimuli can be thought as measurements similar to the pixels of digital images. This is why it may seem very natural for computer vision algorithms to draw parallels with research on visual perception, although historically those two disciplines have been developed relatively independently.

A small number of researchers have tried to formalize the descriptive and purely empirical grouping principles from the Gestalt theory into a more rigorous mathematical theory, that of Lowe [81] and Zhu [141]. The most comprehensive work so far is, to the best of our knowledge, due to Desolneux et al. [47] who developed a numerical framework for detecting "partial gestalts" in digital images. The notion of partial gestalt was introduced by Desolneux et al. to differentiate between global gestalts (or complete objects) from their partial gestalts, or sub-parts, with geometric qualities such that they get combined together by the grouping principles. In other words, angle detection, edge detection, as well as shape detection (lines, circles, etc.), are all algorithms for detecting partial gestalts. Thus, one remarkable aspect of Desolneux et al.'s work is the attempt to cast those basic but diverse image processing problems into a common framework. Let us present it in more details.

The central idea of Desolneux et al.'s work relies on the Helmholtz principle [81], which states that structures having a low probability to appear in noise are instantly perceived:

Definition 1. Helmholtz principle (from Page 31 in [47]):
Whenever some large deviation from randomness occurs, a structure is perceived.

Let us consider an image containing K atomic objects, of which m share a common attribute such as color or position. In order to decide whether the considered aspect (i.e., partial gestalt) will be perceived, it is assumed that the attribute is distributed uniformly at random and independently among the K objects. This is called the "a-priori" random assumption. The Helmholtz principle says that if the observed quality could have happened by chance under the a-priori random assumption, it will not be perceived. But if it was unlikely to appear in noise, the quality will be perceived "a-contrario".

Turning the Helmholtz principle into a computational procedure still requires to define what is meant by "an observed quality happening by chance". Intuitively, an event is unlikely to happen by chance if it has a very small probability to occur. But how to decide if a probability is small enough? Another problem with probabilities pointed out by Desolneux et al. is that in some cases they can be cumbersome to compute; the Birthday paradox is a well-known example [47]. This is why they preferred working with expectations. Expectations can also solve the probability thresholding issue in an elegant manner when using carefully crafted random variables.

Returning to the problem of deciding whether the fact that m out of K objects in an image share a common property is unusual, let us denote by C_m the random variable that counts the number of tuples of m objects that have the considered property, under the a-priori random assumption. This assumption allows us to define $E[C_m]$, the expected value of C_m. Following Desolneux et al.'s terminology, we call $E[C_m]$ the *Number of False Alarms* (NFA) [47]. When NFA is strictly less than 1, then on average, finding m objects with the considered property is unexpected; thus the Helmholtz principle states that the resulting structure will be perceived. In this case, Desolneux et al. call this event *meaningful*. Otherwise, if NFA is greater or equal to 1, it is a "false alarm". One great advantage of the Number of False Alarms is that the natural threshold of 1 has an intuitive theoretical grounding: the event is either expected or unexpected. It also allows derived models to be non-parametric. Obviously, other thresholds may also be chosen if necessary:

Definition 2. ϵ-meaningful events (from Page 38 in [47]):
An event is ϵ-meaningful if the expectation of the number of occurrences of this event is less than ϵ under the a-priori random assumption. When $\epsilon \leq 1$, the event is said to be meaningful.

We shall keep using this terminology in this thesis. Note that Desolneux et al.'s approach is related to the traditional framework of statistical hypothesis testing (e.g., tests for randomness). Indeed, it can be seen as a statistical test where the null hypothesis assumes the considered property is distributed uniformly at random and thus that it is not perceived. The null hypothesis is rejected when ϵ-meaningful events are observed; said differently, the decision criterion is ϵ rather than some significance level. The relation with hypothesis testing is discussed at length in Chapter 15 of [47]. Also, for concrete applications to computer vision problems, the reader may refer to [47, 66, 96].

In the next section, we explain how these notions can be adapted to text mining.

3.2 Helmholtz principle for data and text mining

Despite their computer vision origin, the notions of *Number of False Alarms* and ϵ*-meaningful events* presented in the previous section are extremely generic. Noting this fact, Balinsky et al. extended these notions to data mining using a simple model [22] that can be described in terms of the classical balls-and-bins setting.

3.2.1 NFA for m-tuples in a balls-and-bins experiment

Let us consider a set of K balls thrown uniformly and independently at random into N bins, as illustrated in Figure 3.2. Let us also denote by C_m the random variable that counts the number of m-tuples of balls found in the bins at the end of the experiment, where $m \in \{1, 2, \ldots, K\}$. Without loss of generality, each of the K balls is initially assigned a unique index between 1 and K. Therefore, each m-tuple can be defined by the increasing sequence of indices $(i_1, i_2, \ldots, i_m)$ of the balls in the tuple.

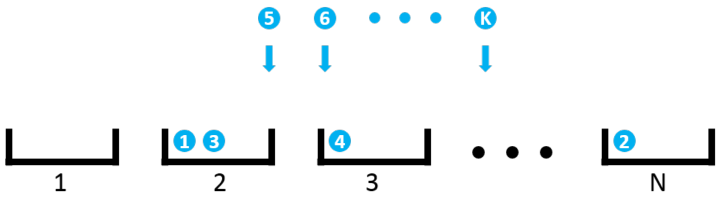


Figure 3.2: Illustration of the balls-and-bins experiment, with K balls thrown uniformly at random into N bins.

An interesting question to answer in this setting is whether it is expected to find an m-tuple of balls in one of the N bins. Following the definition given in the last section, the Number of False Alarms of an m-tuple is the expected value of C_m:

$$NFA_{B\&B}(m, K, N) = E[C_m]. \tag{3.1}$$

In order to compute $E[C_m]$, let us introduce the indicator random variable $\chi_{(i_1,\ldots,i_m)}$

denoting whether all the balls of the m-tuple $(i_1, \ldots, i_m)$ land in the same bin.

$$\chi_{(i_1,\ldots,i_m)} = \begin{cases} 1 & \text{if all } m \text{ balls with indices } (i_1,\ldots,i_m) \text{ land in the same bin,} \\ 0 & \text{otherwise.} \end{cases} \tag{3.2}$$

By definition of C_m, we have:

$$C_m = \sum_{\substack{(i_1,\ldots,i_m): \\ 1 \le i_1 < \cdots < i_m \le K}} \chi_{(i_1,\ldots,i_m)} \tag{3.3}$$

and by the linearity of expectation:

$$E[C_m] = \sum_{\substack{(i_1,\ldots,i_m): \\ 1 \le i_1 < \cdots < i_m \le K}} E[\chi_{(i_1,\ldots,i_m)}] = \sum_{\substack{(i_1,\ldots,i_m): \\ 1 \le i_1 < \cdots < i_m \le K}} P(\chi_{(i_1,\ldots,i_m)} = 1). \tag{3.4}$$

The probability that all balls in a given m-tuple land in the same bin, given the independence assumption, is:

$$P(\chi_{(i_1,\ldots,i_m)} = 1) = 1 \times \frac{1}{N} \times \cdots \times \frac{1}{N} = \frac{1}{N^{m-1}}. \tag{3.5}$$

The number of possible m-tuples is $\binom{K}{m}$, so Equation (3.4) becomes:

$$E[C_m] = \sum_{\substack{(i_1,\ldots,i_m): \\ 1 \le i_1 < \cdots < i_m \le K}} \frac{1}{N^{m-1}} = \binom{K}{m} \frac{1}{N^{m-1}}. \tag{3.6}$$

Finally, from Equations (3.1) and (3.6), unexpected events (i.e., unusually high numbers of balls in a bin) are such that:

$$NFA_{B\&B}(m, K, N) = \binom{K}{m} \frac{1}{N^{m-1}} < 1. \tag{3.7}$$

In the next section, the relation of this balls-and-bins model to text mining, and more precisely to keywords extraction, will be explained.

3.2.2 Practical adaptation of the balls-and-bins model to keywords extraction

First and foremost, one should precise the notion of *keyword*. It seems intuitive that the frequency of a word should be correlated with its importance. However, frequencies alone are not sufficient as stop-words are frequent but unimportant. Indeed, stop-words are expected to appear everywhere with approximately the same frequency. On the contrary, new topics emerge with the word frequencies from related vocabulary rising sharply. Therefore, the context also plays an important role. A common definition for a keyword is a "word with a local and unusual rise in frequency" [22, 70].

Coming back to the balls-and-bins model, balls represent words and bins represent the local contexts used for defining such unusual rises in frequency. We shall call generically those bins "containers". Thus, an unusually high number of occurrences of a word inside a container is what defines a keyword. The practical definition given to containers will depend on the data at hand and the application. Common examples are:

- each paragraph (or section, or chapter, etc.) from a document,
- each document from a collection of documents, or
- each class from a categorized corpus of documents.

However, containers may actually be any structure of interest. In some cases, no obvious structure is available; for example a document may not be divided into paragraphs. In such cases, one could use overlapping containers, for example sliding windows of a fixed size (in number of words or number of sentences).

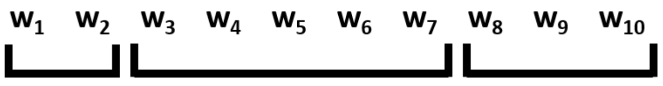
Model refinements

The model presented so far is not realistic as it assumes that the probability of a word to land in any of the N containers is the same for every container: $1/N$. In practice, containers mirror real structures from the textual data such as paragraphs in a document, so they will be of varying sizes. Let us define the length of a container to be the number of words it contains. The model can be easily adapted by using an *adaptive* number of containers N_{adapt} taking into account the length of the container currently considered:

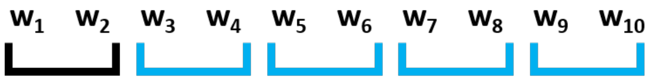
$$N_{adapt} = \left\lfloor \frac{L_D}{L_C} \right\rfloor \tag{3.8}$$

where L_D and L_C are respectively the lengths, in words, of the entire collection of containers and of the current container. $\lfloor x \rfloor$ denotes the integer part of x.

Thus, each container is associated a different N_{adapt} which does not, in most cases, represent real containers. This is why working with overlapping containers, such as sliding windows, is not a problem: indeed, N_{adapt} only depends on L_D and L_C. Let us consider a document made of 10 words in total and assume this document contains three paragraphs of 2, 5 and 3 words respectively, as illustrated in Figure 3.3a. For each paragraph, the NFA computations will set, respectively, N_{adapt} to $\lfloor \frac{10}{2} \rfloor = 5$, $\lfloor \frac{10}{5} \rfloor = 2$ and $\lfloor \frac{10}{3} \rfloor = 3$ (see Figure 3.3b).


w_1 w_2 w_3 w_4 w_5 w_6 w_7 w_8 w_9 w_{10}

(a) Real containers (e.g., paragraphs) based on the structure of a document.


w_1 w_2 w_3 w_4 w_5 w_6 w_7 w_8 w_9 w_{10}

(b) Fictitious containers used for all NFA computations regarding the first container.

Figure 3.3: Using the adaptive number of containers N_{adapt} to cope with containers of varying lengths.

Putting it all together, the Number of False Alarms for a word in a container is:

$$NFA_{keyword}(m, K, N_{adapt}) = \binom{K}{m} \frac{1}{N_{adapt}^{m-1}}, \tag{3.9}$$

where m is the frequency of the word in the container, K the frequency in the collection of containers and N_{adapt} is computed using Equation (3.8). And this word is a keyword (i.e., its frequency is unusually high in the container) if:

$$NFA_{keyword}(m, K, N_{adapt}) < 1. \tag{3.10}$$

Meaningfulness scores

The Number of False Alarms is a strictly positive number that represents an expected number of occurrences. NFA values in the $(0, 1)$ open interval represent unexpected events and the smaller the NFA value, the more unexpected the event (or, in other words, the more important the keyword). This may sound counter-intuitive and one might prefer working with a measure of importance where larger values represent more important keywords.

To this end and as suggested in [22], we prefer to work with a *meaningfulness score*, denoted by $meaning$, that will be used heavily in the remaining of this thesis:

$$\begin{aligned} meaning(m, K, N_{adapt}) &= -\frac{1}{m} \log NFA(m, K, N_{adapt}) \\ &= -\frac{1}{m} \log \left[\binom{K}{m} \frac{1}{N_{adapt}^{m-1}} \right]. \end{aligned} \tag{3.11}$$

There are several theoretical reasons for such a transformation. Desolneux et al. use it as a unified format for closed form estimation purposes, as large values of the parameters m, K and N_{adapt} may lead to computational issues (see Chapter 4 in [47] for details). Based on theoretical physics, Balinsky et al. [22] also argue that $meaning$ represents an "energy per particle" (i.e., an occurrence of a word), allowing comparisons between different physical systems (i.e., collections of containers).

Regarding computations, it is important to note that in Equation (3.11), m and K are potentially very large numbers, so $\binom{K}{m}$ can take arbitrary large values. To avoid overflow issues and speed up computations, the $meaning$ formula can be simplified as:

$$\begin{aligned} meaning(m, K, N_{adapt}) &= -\frac{1}{m} \log \left[\binom{K}{m} \frac{1}{N_{adapt}^{m-1}} \right] \\ &= -\frac{1}{m} \left[\log \binom{K}{m} - \log N_{adapt}^{m-1} \right] \\ &= -\frac{1}{m} \left[\log \frac{K \ldots (K-m+1)}{m!} - (m-1) \log N_{adapt} \right] \\ &= -\frac{1}{m} \left[\sum_{i=K-m+1}^{K} \log i - \sum_{i=1}^{m} \log i - (m-1) \log N_{adapt} \right]. \end{aligned}$$

Also, for clarity and convenience reasons, we will often use the equivalent set of parameters $meaning(w, P, D)$, where w is a word that appears K times in the container P from the collection of containers D.

In summary, the meaningfulness score $meaning(w, P, D)$ quantifies the importance of the word w in the container P (e.g., a paragraph), relative to the collection of containers D (e.g., the associated document). Positive meaningfulness scores correspond to keywords and the higher the $meaning$ score, the more important the keyword. Negative meaningfulness scores correspond to non-meaningful words (i.e., words that are less important).

In the next chapter, we investigate a first concrete application: information retrieval.

Chapter 4

Information Retrieval and Enterprise Search

Information retrieval refers to the theory and algorithms behind search engines. More formally, it is the process of retrieving the documents that are relevant to an information need, from a given collection of documents. The word "document" in this context is extremely general and can refer to text, audio, video, etc. as well as, increasingly, combinations of such types. However, we will remain focused in this thesis on textual systems.

Information needs are described by user-defined *queries*, generally consisting of one or more words. The objective of a retrieval system is to identify the subset of documents that is relevant to the information need expressed by the query and usually rank these documents by decreasing order of relevance. Information retrieval is crucial in the current web era but it is far from a new research area. Structuring and organizing collections of documents is an immemorial need that started with ancient libraries. Computers have greatly simplified this process and naturally, information retrieval has been a large research area starting from the second-half of the 20th century.

In this chapter, we investigate enterprise search, which is information retrieval geared towards collections of documents found on companies' intranets. More precisely, a new term weighting scheme for ranking documents by order of relevance is developed and a complete search engine for enterprise search is built upon it. We start with a review of the state-of-the-art in Section 4.1. Our algorithm and full system are described in Section 4.2. The evaluation methodology is introduced in Section 4.3 while Section 4.4 presents and analyzes the results. Finally, Section 4.5 concludes the chapter.

A brief description of the fully-fledged search engine that has been developed as part of this work is available in Appendix A.1, page 106.

4.1 A brief literature review

4.1.1 Indexing

The first step when building a retrieval system is to index the collection of documents that is to be searchable. An *index* is a data structure that allows fast retrieval of the documents that contain the query terms. The most common type of index is the so-called inverted index that stores, for each unique term in the collection of documents, the list of all documents containing this term, as illustrated in Table 4.1. The example inverted index allows to find directly that the query "picture" should result in two documents being returned to the user: Doc$_2$ and Doc$_3$. Without such an index, one would need to do an expensive linear scan through the entire collection of documents. It should be noted that the choice of index terms is crucial as words that are not indexed cannot be retrieved. As an example, if stop-words filtering is done before indexing, the queries "to be or not to be" and "The Who" (a rock band) will not return any results since they contains only stop-words.

Table 4.1: Building a simple inverted index. In this example, stop-words are not indexed.

(a) An example collection of documents

Document ID	Document text
Doc$_1$	To be or not to be, that is the question.
Doc$_2$	A picture is worth a thousand words.
Doc$_3$	Every picture tells a story.

(b) The corresponding inverted index

Term	Document IDs where term appears
picture	Doc$_2$, Doc$_3$
question	Doc$_1$
story	Doc$_3$
tells	Doc$_3$
thousand	Doc$_2$
words	Doc$_2$
worth	Doc$_2$

Historically, index terms used to be manually assigned by experts to each document. However, the vast majority of search engines work nowadays with full-text indices. Full-text indexing simply means that all terms are stored in the index after preprocessing. As such, our Table 4.1b example is a full-text index, even if stop-words are removed. As already pointed out, the preprocessing techniques used (stop-words removal, normalization, stemming, etc.) have a large impact on the end results and thus need to be carefully devised.

But more than just retrieving potentially relevant documents by means of indexing, modern systems need to rank these documents by decreasing order of relevance. Indeed, the results that users first see should be the documents that are most likely to be relevant. Several retrieval models have been developed over the years to do such a ranking. Let us present the most popular ones.

4.1.2 Retrieval models

Early information retrieval systems relied on *set-theoretic models*, which represent both documents and queries as sets of terms. The best known example is the *standard Boolean model* (see Chapter 1 in [86]), in which queries are written as Boolean expressions. Here is an example of Boolean query:

(*picture* OR *image*) AND (*words*) AND (NOT *story*)

Unfortunately this model does not provide a ranking mechanism: a document either does or does not match the query. Extensions have been proposed to introduce document rankings. The most notable examples include fuzzy models [126] where fuzzy set theory allows terms to have varying degrees of membership to a document and the extended Boolean model [119] which draws ideas from algebraic models.

Algebraic models are arguably the most influential type of models as they appear in one way or another in almost all current retrieval systems. In the *standard vector space model*, both documents and queries are represented as vectors in a Euclidean space where each dimension corresponds to an element of the vocabulary (i.e., a term). The similarity between a document D and a query Q, respectively represented by the vectors $\mathbf{d}$ and $\mathbf{q}$, can be computed using the cosine similarity:

$$\cos(\mathbf{d}, \mathbf{q}) = \frac{\mathbf{d} \bullet \mathbf{q}}{\|\mathbf{d}\| \times \|\mathbf{q}\|} \tag{4.1}$$

Intuitively, the cosine is a function of the angle between the two parameter vectors $\mathbf{d}$ and $\mathbf{q}$, with high cosine values representing vectors with similar directions. But we have not yet defined the vector weights; how should a preprocessed document be transformed into a vector to reflect the importance of each term in this document? This question has received huge attention from researchers and TF-IDF term weighting schemes [118] have become ubiquitous. TF-IDF schemes have little theoretical grounding; the first serious attempt to justify them was provided by Robertson in 2004 [111], several decades after they were first introduced. However, they have become widely popular since they are both intuitive and very effective in practice. TF stands for *Term Frequency* and accounts for the idea that the more a term appears in a document, the more this term is likely to be important inside this document. Many variations of TF exists but a common normalized version is:

$$\mathrm{TF}(t, D) = \frac{f_{t,D}}{\max\limits_{u \in D} f_{u,D}} \tag{4.2}$$

where t denotes a term and $f_{t,D}$ the frequency of t inside the document D. However, terms that are very frequent in the collection of documents, including stop-words, are usually less informative than rarer terms. IDF, which stands for *Inverse Document Frequency*, reflects this idea. Again, many variations exist but a popular one is:

$$\mathrm{IDF}(t, C) = \log \frac{|C|}{DF_t} = \log \frac{|C|}{|\{D \in C : t \in D\}|} \tag{4.3}$$

where C denotes the collection of documents and DF_t the document frequency of t, that is the number of documents from C where t occurs. Thus the numerator is the total number of documents in the collection and the denominator the number of documents which contain t. Then, TF-IDF weights are simply the products of TF and IDF components, thus balancing their respective importance criteria. In summary, vector space models usually work as follows:

- each document D is represented as a TF-IDF vector $\mathbf{d}$, where the weight for each dimension (i.e., term) is computed as $\text{TF-IDF}(t, D, C) = \mathrm{TF}(t, D) \times \mathrm{IDF}(t, C)$,
- when a user types a query Q, the query string is converted to a query vector $\mathbf{q}$ that is often a simple binary incidence vector denoting terms appearing in the query or a vector made of TF weights,
- for each document in the collection where at least one query term appears, a relevance score $score_{\text{TF-IDF}}(D, Q)$ is computed using Equation (4.1) and these documents are returned in decreasing order of relevance. This step can be achieved very quickly thanks to the inverted index.

It should be pointed out that dozens of TF-IDF variants have been proposed over the years. Salton and Buckley summarize and formalize the most prevalent ones in [118]. The notion of "fields" has been introduced more recently to differentiate between the different document sub-parts and thus give more emphasis to the most important sub-parts, like the title [137].

These vector space models are so popular because of their effectiveness, simplicity and scalability. They have a number of issues though. First, dimensions are obviously not orthogonal in practice due to synonymy, antonymy and other relations between words. Second, they are less expressive than Boolean models as, for example, they do not have an equivalent to the Boolean negation (i.e., the NOT operator). Also, similarly to set-theoretic models, they are a bag-of-words approach in which the order of words is ignored, so semantic and syntactic relations between words are lost. Alternatives that solve some of these issues have been proposed, most notably the generalized vector space model [136] which relaxes the orthogonality assumptions and Latent Semantic Indexing (LSI) [46] that relies on Singular Value Decomposition to work with dimensions corresponding to semantic concepts rather than just terms.

The last major type of models are *probabilistic models*. The original motivation for developing probabilistic models was mainly to improve the theoretical grounding of the empirical TF-IDF models. Many probabilistic approaches have been proposed. One of the most influential is the Binary Independence Model [115] which has led to one of the current state-of-the-art ranking scheme called BM25 [112, 113]:

$$BM25(t, D, C) = \frac{f_{t,D}(k_1 + 1)}{f_{t,D} + k_1(1 - b + b\frac{|D|}{avgdl})} \times \log \frac{N - DF_t + 0.5}{DF_t + 0.5} \tag{4.4}$$

$$score_{BM25}(D, Q) = \sum_{t \in Q} BM25(t, D, Q) \tag{4.5}$$

where k_1 and b are tuning parameters, $avgdl$ is the average document length (in number of terms) in the collection of documents and N the total number of documents in the collection. The derivation of this formula is rather lengthy and can be found in [113]. It is interesting to note how the BM25 formula remains clearly inspired by algebraic models: the first part can be considered as a TF component and the second part as an IDF component. Alternative probabilistic models, that show good performance but do not try to model relevance directly, include language modelling approaches [106] and divergence from randomness models [18].

4.1.3 Learning-to-Rank

Over the years, initial ranking functions such as TF-IDF and BM25 have proven their efficiency and robustness for a variety of retrieval tasks. But they also have clear limitations. First, they make simplifying assumptions when representing documents; for example they ignore the proximity of query terms in the documents. Second, they do not take into account many potentially useful relevance features. For example, one might prefer to have well-written and recent documents at the top of the search results.

Supervised learning can be used to alleviate these problems. More precisely, *learning-to-rank* algorithms can be used to automatically learn a good ranking function based on a set of queries with gold-standard relevance judgements. Learning-to-rank allows the use of many features and can thus incorporate a variety of aspects that affect document relevance.

Given the large computational requirements of such algorithms, retrieval systems that use learning-to-rank usually work in a two-stage process. First, a fast ranking function like TF-IDF is used to retrieve an initial ranking of relevant documents. As an illustration, this ordered list could be:

$$initial\_ranking(Q) = [Doc_a, Doc_b, Doc_c, Doc_d, Doc_e].$$

Then, the learning-to-rank algorithm re-ranks a fixed number of top documents (say, 50) to ensure the most relevant ones are placed at the very top of the results list. This re-ranking function can take into account many features ignored by the initial ranking function, such as the proximity of query terms inside the document, the document quality, etc. in order to output the best permutation possible:

$$\begin{gathered} top\_documents(Q, 3) = [Doc_a, Doc_b, Doc_c] \\ best\_permutation(Q, [Doc_a, Doc_b, Doc_c]) = [Doc_b, Doc_c, Doc_a]. \end{gathered}$$

The re-ranked top documents are output first, followed by the remaining tail of the initial list of documents:

$$final\_ranking(Q) = [Doc_b, Doc_c, Doc_a] + [Doc_d, Doc_e].$$

Three main learning-to-rank approaches exist [74]:

- *pointwise algorithms*, that use traditional classification or regression methods to directly predict the relevance score of a document,

- *pairwise algorithms*, building a binary classifier that predicts the more relevant document for each pair of documents and then reconstructing a ranked list of documents, and
- *listwise algorithms*, that directly optimize for retrieval performance measures like nDCG (which we will discuss in more details in the evaluation section of this section). This is difficult because such metrics are not continuous.

We use pairwise and listwise algorithms in this thesis. However, we will mainly treat learning algorithms as blackboxes as these are not the primary focus of our work.

4.2 Proposed algorithm

In this section, we propose the use of the Helmholtz principle to quantify relevance in an information retrieval context. We also outline a search engine developed for enterprise search, building upon the proposed term weighting scheme and augmenting it with learning-to-rank capabilities.

4.2.1 Preprocessing

It goes without saying that the collection of documents must be preprocessed at indexing time in exactly the same way as the query at search time. This is because the query terms must exactly match the same document terms in the index.

A very simple preprocessing pipeline is used: the input text, either a document or a query, is lower-cased and split on every non-alphabetical character. The resulting tokens are then stemmed with the Porter stemming algorithm.

Two important details are worth highlighting. First, stemming is very simple way to fuzzily match query terms with document terms. Indeed, a search for "cat" will retrieve documents containing the word "cats" as one might expect. As always, a trade-off is necessarily made: this process increases recall (more relevant documents will be retrieved) at the cost of lower precision (some less relevant documents will also be retrieved). Second, it might be very tempting to remove stop-words, in order to reduce the size of the index but also to increase precision. We choose to not filter stop-words because this has a number of negative consequences on search, for reasons already mentioned.

4.2.2 Term weighting scheme

Most of the current state-of-the-art weighting schemes are either empirically derived, such as TF-IDF, or arise from lengthy and complex derivations, such as BM25. The measure of meaningfulness $meaning(w, D, C)$, which stems from the Helmholtz principle and was presented on Equation (3.11) page 22, provides a simple and theoretically sound measure of word importance. We propose its use for term weighting in an information retrieval context, where it can provide a simple measure of importance for the word w inside the document D relative to the entire collection of documents C. Using the notations introduced in Section 4.1, Equation (3.11), from page 22, can be rewritten as:

$$meaning(t, D, C) = -\frac{1}{f_{t,D}} \log \left[\binom{f_{t,C}}{f_{t,D}} \frac{1}{N_{adapt}^{f_{t,D}-1}} \right] \tag{4.6}$$

where $f_{t,D}$ and $f_{t,C}$ are the frequencies of the term t inside the document D and collection of documents C. N_{adapt} is set to $|C|/|D|$, that is the length (in number of terms) of the collection of documents divided by the length of the document (also in number of terms).

Terms that do not appear in a document are assumed to have a $meaning$ value of 0. Therefore, terms with a negative meaningfulness score are truncated at 0, otherwise these negative values would act as penalties on relevance scores. Thus, the $meaning$ formula becomes in the search context:

$$meaning_{search}(t, D, C) = \begin{cases} meaning(t, D, C) & \text{if } t \in D \;\&\; meaning(t, D, C) > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{4.7}$$

Finally, inspired by algebraic models, we define the relevance of a document D to a query Q simply as the sum of the meaningfulness scores of the query terms it contains:

$$score_{meaning}(D, Q) = \sum_{t \in Q} meaning_{search}(t, D, C) \tag{4.8}$$

Note that documents containing only non-meaningful query terms are assigned zero relevance scores similarly to documents that do not contain any query terms. This is not a problem in practice since the latter documents, which are entirely non-relevant, are not retrieved from the inverted index and thus are never part of the actual computations.

Algorithm 4.1 gives the full pseudo code for initial ranking computations.

Algorithm 4.1: Initial ranking computations.

```
Initialize $relevant\_docs$ to the empty list.
Preprocess the query $Q$ into a list of terms.
For each term $t$ in the query:
    Get all documents containing $t$ (from the inverted index).
    For each such document $D$:
        If $D$ is not in $relevant\_docs$:
            Add $D$ to $relevant\_docs$, with $score_{meaning}(D,Q) = 0$.
        Compute $meaning_{search}(t,D,Q)$ [see Equation (4.7)].
        Add $meaning_{search}(t,D,Q)$ to $score_{meaning}(D,Q)$.
Sort $relevant\_docs$ by decreasing order of $score_{meaning}(D,Q)$.
Return $relevant\_docs$.
```

4.2.3 Augmenting the system with Learning-to-Rank

A search engine could rely solely on the term weighting mechanism introduced in the last section and directly return the initial ranking of documents to the user. However, in order to improve these document rankings based on a richer and more powerful set of features, we augment our system with learning-to-rank. In this section, we describe the features that we designed for the system and explain how learning-to-rank algorithms are trained.

It is worth pointing out that the search engine developed as part of the work for this thesis, as well as the experiments carried in this chapter, are aimed at enterprise search. Despite being similar with web search, enterprise search differs on a number of points and especially in the choice of features for learning-to-rank: for example, web search benefits from features detecting the techniques used by websites to artificially get better rankings, known as "spamdexing" [56]. No such adversarial behaviour is to be expected in the enterprise search context. Also, document metadata (author, creation and modification dates, etc.) should be more consistently available in enterprise collections. These differences are reflected in our choice of features.

Features design

Learning-to-rank features are usually divided in two main types: query-dependent and query-independent features [83]. We prefer to use slightly more precise groupings: query-document features, document features and query features. Let us present each of these types in turn.

Query-document features Query-document features attempt to quantify how well a document matches the query. Features based on initial ranking formulae such as TF-IDF or BM25 are obviously common [109], as well as various other measures indicating how many query terms actually appear in a document or how close these query terms appear to each other. The query-document features used in our models are presented in Table 4.2.

Table 4.2: Query-document features.

Feature	Description
Meaningfulness scores sum	Sum of $meaning_{search}$ scores for the document, over all query terms. Query terms that do not appear in the document are assigned 0 scores.
Meaningfulness scores average	Average of $meaning_{search}$ scores for the document, over all query terms.
Meaningfulness scores std deviation	Standard deviation of $meaning_{search}$ scores for the document, over all query terms.
Nb covered query terms in title	Number of unique query terms that appear in the title of the document.
Nb covered query terms in content	Number of unique query terms that appear in the body of the document.
Minimum pairwise distance in title	Minimum pairwise distance, over all pairs of query terms, in the title of the document.
Minimum pairwise distance in content	Minimum pairwise distance, over all pairs of query terms, in the body of the document.
Average pairwise distance in title	Average pairwise distance, over all pairs of query terms, in the title of the document.
Average pairwise distance in content	Average pairwise distance, over all pairs of query terms, in the body of the document.
Title min cover	Minimum coverage, in the title of the document, of all query terms.
Content min cover	Minimum coverage, in the body of the document, of all query terms.

The definitions and purposes of most features in the table should be clear, apart from minimum coverage and pairwise distance [127] that were not introduced. These are proximity measures quantifying how close the query terms appear together in the document. They reflect the idea that query terms should co-occur closely in documents. Indeed, in response to the query "gray cat", a document containing "the gray cat" should be considered more relevant than a document containing "the gray wolf and the red cat".

Minimum coverage is defined as "the length of the shortest document segment that covers each query terms at least once in a document" [127]. With the previous example, the first document has a minimum coverage of 2 while the second document has a minimum coverage of 6.

The *pairwise distance*, defined between any two (unique) query terms, is the smallest distance between these two terms inside a document of the collection. As there is a pairwise distance for each pair of unique query terms, these distances are then be combined with an aggregation operator such as the minimum or average. For illustration purposes, the pairwise distances for the terms "gray" and "cat" are, respectively, 1 and 5 in the first and second document from our running example.

In case a query term does not appear in a document, these proximity measures are considered by default to be equal to the length of the document. Thus, these two proximity measures achieve similar goals. Minimum coverage is oriented towards matching exactly the entire query, while pairwise distance allows some query terms to be missing so it is more robust in case synonyms are used or only partial matches occur.

As a final remark, it is worth noting that we differentiate the title from the remaining content of the document for many features. This is because occurrences of query terms in the title are often good indicators of relevance, more so that occurrences in the remaining body of the document. This is a very common trick in information retrieval, that has been recently extended to the notion of "fields" (for example: title, main content, anchors and comments) that are scored independently [114]. Such fields could be integrated in our search engine in a trivial way, with more features; however, these are necessarily corpus specific so we avoid them in the interest of genericness.

Document features Document features are indicative of the quality of a document. Assuming an equivalent relevance, the better-written document should appear in the search results before the equivalent badly-written document. The document features used in our models are presented in Table 4.3. Note that these features can be pre-computed at indexing time and stored alongside the index, to avoid repeated calculations and return slightly faster results at query time.

Table 4.3: Document features.

Feature	Description
Corpus	Binary features (one for each corpus) indicating the source of the document, if several corpora (sources) of data are indexed
Document length	Document length in number of terms
Title length	Title length in number of terms
Content length	Length of document content in number of terms
Creation age	Time in days since document creation
Modification age	Time in days since last document modification
Average rating	Average user ratings, on a scale 1 to 5 stars (default 2.5)
Nb ratings	Number of user ratings
Nb ingoing links	Number of in-going links
Nb outgoing links	Number of out-going links

Document features should capture several aspects of document quality. Quality is unfortunately a rather subjective notion and therefore difficult to quantify. Many complex measures have been proposed, for example taking into account document structure [63], but we focus instead on generic features that can be extracted from new document collections with no or little development efforts. Let us briefly mention the rationale behind our choice of features:

- **document corpora:** some corpora/sources are better than others (e.g., Wikipedia [16] is a generally reliable source on the internet).
- **document age:** the contribution of document age will be very domain specific; in some applications, documents become quickly irrelevant (e.g., news stories) while in collaborative environments, older documents may be more comprehensive, fact-checked and/or polished.
- **ingoing and outgoing links:** documents referenced by many other documents are popular and thus probably of good quality. Many complex algorithms working with links have been proposed, such as PageRank [102], but for simplicity we limit ourselves to counts.
- **user feedback:** users can provide, directly or indirectly, a lot of valuable information on documents they like or dislike. Our search engine provides several ways for them to leave feedback, so we incorporate some of this feedback in the features for ranking.

Query features Finally, query features allow learning-to-rank models to specialize for different kinds of queries [83]. For instance, let us consider features based on pairwise distances. Since these features take into account pairs of query terms, they are only useful for queries made of several words. So when the query length (in number of terms) is also provided as a feature, a learning-to-rank algorithm can learn to discard pairwise distance features for queries made of a single term.

We use only a few query features in our models; these are presented in Table 4.4.

Table 4.4: Query features.

Feature	Description
Query length	Query length in number of stems
Nb unique words	Number of unique query words (before stemming)
Nb unique stems	Number of unique query stems

Ranking algorithms

Regarding learning-to-rank algorithms, we experiment with both pairwise and listwise approaches.

A few open-source implementations of pairwise algorithms exist, such as *SVMRank* [10, 64]. However, we preferred to re-implement the pairwise transformation process from scratch to allow experiments with any binary classifier. Indeed the pairwise approach estimates, for each pair of top documents D_1 and D_2, which document is more relevant than the other relative to a query Q. To do so, it can take as input the concatenation of the two documents' feature vectors $feature(D_1, Q)$ and $feature(D_2, Q)$, outputting $+1$ if D_1 is more relevant than D_2 or -1 otherwise. This is a standard supervised learning problem and as such, any binary classifier can be used to do so (logistic regression, SVM, random forest, etc. [9]). Note that when concatenating the two feature vectors, there is no need to duplicate the query features because these are the same for every documents, given a query. Once every pair of documents has been classified, a new ranking is constituted using a simple voting procedure: the document most often considered as more relevant is placed at the top of the results list, and so on. In this setting, it is important to note that training instances are not independent from each other. The reason is two-fold: first, the same document participates in several pairs, but the query structure of the dataset also means that documents relating to the same query are dependent on each other. As such, the pairwise setting violates the assumption that underpins most classification algorithms, namely that the data are Independent and

Identically Distributed (IID). However, this is not an issue in practice as traditional classifiers still yield good performance.

Listwise approaches, since they work directly with the full list of top documents, are simpler to use in the sense that they do not require the documents' feature vectors to be transformed. Implementing such algorithms is however more complex, so the RankLib library [8] is used. Note that even though training instances (i.e., queries) are now IID, training a classifier in this setting is harder since the number of training instances for the same dataset is much smaller.

With both approaches, all features are standardized: each feature is normalized by substracting the mean and dividing by the standard deviation of the corresponding feature. The means and standard deviations are estimated on the training set. Such a normalization procedure is applied because some machine learning algorithms are sensitive to the scale of features, in terms of re-ranking performance and/or training times. Also, constant features are filtered. This can happen, say, when no user feedback is available, which is the case with the evaluation datasets used later in the chapter.

4.3 Evaluation methodology

4.3.1 Evaluation procedure

Our proposed term weighting scheme based on meaningfulness scores is evaluated and compared against the traditional TF-IDF and BM25 schemes.

There are many variants of TF-IDF so we use the most prevalent one [86], which is slightly more elaborate than the one presented in the review (see page 26). It is denoted LNC-LTC using the standard SMART notation [117] and is equal to:

$$score_{LNC-LTC}(D,Q) = \sum_{t \in D \cap Q} \frac{1+\log(f_{t,D})}{\sqrt{\sum_{t \in D} \Big(1+\log(f_{t,D})\Big)^2}} \times \frac{(1+\log(f_{t,Q})) \times \log(N/DF_t)}{\sqrt{\sum_{t \in Q} \Big((1+\log f_{t,Q}) \times \log(N/DF_t)\Big)^2}} \tag{4.9}$$

where D and Q represent respectively the document and the query, $f_{t,D}$ is the frequency of the term t in D, N is the total number of documents in the collection and DF_t is the document frequency of t.

The BM25 measure is defined as presented in Equation (4.5), page 27. The k_1 and b tuning parameters are set, respectively, to 1.6 and 0.75 according to popular default values (see Section 11.4.3 in [86]).

Regarding learning-to-rank, experiments are conducted with several re-ranking algorithms. The effect of the number of documents re-ranked on retrieval performance is also studied. But let us clarify first how retrieval performance is quantified.

4.3.2 Evaluation metrics

Many metrics to evaluate the performance of information retrieval systems exist. Precision, recall and F-score are well-known examples but are usually avoided since they work with sets and as such ignore rankings. We use three popular alternatives, namely Reciprocal Rank (RR), Average Precision (AP) and Normalized Discounted Cumulative Gain ($nDCG$).

Reciprocal Rank (RR) is a simple measure that focuses only on the first relevant document. For a given query Q, it is equal to the inverse of the rank of this first relevant document. Denoting this rank $Rank_{first}(Q)$, we have:

$$RR(Q) = \frac{1}{Rank_{first}(Q)} \tag{4.10}$$

Average Precision (AP) considers all relevant documents in the results list. It is equal to the average precision at rank, over the ranks of all relevant documents that are returned:

$$AP(Q) = \frac{\sum_{i=1}^{|Results(Q)|} Precision@i(Q) \times IsRelevant(Q, i)}{|RelevantDocuments(Q)|} \tag{4.11}$$

where:

- $Results(Q)$ is the set of results returned by the search engine,
- $Precision@i(Q)$ is the precision at rank i (i.e., the ratio of relevant results in the first i results),
- $IsRelevant(Q, i)$ is an binary indicator function, with value 1 if the i-th document is relevant and 0 otherwise, and

- $RelevantDocuments(Q)$ is the set of relevant documents for the query Q.

AP is actually an approximation of the area under precision-recall curve, where data points are (precision, recall) pairs at every rank. Since we want both precision and recall to be high, higher values indicate better document rankings. As we are usually interested in the top results only, an Average Precision at rank k variant exists, which truncates the results list at k results:

$$AP@k(Q) = \frac{\sum_{i=1}^{k} Precision@i(Q) \times IsRelevant(Q, i)}{\min(k, |RelevantDocuments(Q)|)} \tag{4.12}$$

Finally, the Normalized Discounted Cumulative Gain ($nDCG$) quantifies the "gain" (i.e., usefulness) of each document in the results list. Because lower ranks are less useful that high ranks, these gains are logarithmically discounted (i.e., penalized) as the ranks increase:

$$DCG(Q) = Relevance(Q, 1) + \sum_{i=2}^{|Results(Q)|} Relevance(Q, i)/\log_2(i) \tag{4.13}$$

$$nDCG(Q) = \frac{DCG(Q)}{IDCG(Q)} \tag{4.14}$$

where $Relevance(Q, i)$ denotes the relevance level of the i-th result. $IDCG(Q)$ stands for Ideal Discounted Cumulative Gain and is just a normalization factor that is computed using the $DCG(Q)$ formula while assuming a perfect ranking of the search results. Note that in most datasets, relevance levels are not binary (i.e., some documents are more relevant than others), so $nDCG$ has the advantage over AP that it takes into account relevance levels. Similarly to $AP@k$, $nDCG@k$ also exists; it is computed in exactly the same way, except that the results lists (both the one returned by the search engine and the ideal one) are cut-off at k documents.

All these measures are defined for a given query Q. In practice, ranking algorithms are evaluated over many queries, so all the above metrics are averaged. Reciprocal Rank RR simply becomes Mean Reciprocal Rank:

$$MRR = \frac{\sum_{i=1}^{|Queries|} RR(Q_i)}{|Queries|}. \tag{4.15}$$

We define similarly MAP (Mean Average Precision), $MAP@k$, Mean $nDCG$ and

Mean $nDCG@k$. Finally, note that all these measures (both the single query and averaged versions) take values in the $[0, 1]$ range, with higher values corresponding to better performance.

4.4 Experimental results on the TREC datasets

4.4.1 The CERC collection and associated TREC datasets

The evaluation of information retrieval algorithms requires datasets of queries and associated gold-standard relevance judgements, that is to say, a list of relevant documents for each query.

Evaluating and comparing algorithms designed for enterprise search is difficult for several reasons. First, the creation of a public and realistic dataset is complicated because enterprise data are in large parts confidential. Second, even if enterprise data are available for a given project, creating gold-standard judgements allowing to evaluate and fine tune search algorithms is a costly and time-consuming task. Alternatively, gold-standard judgements can be derived from large quantities of search logs [64]. However, this approach requires a search engine to be used without a learning-to-rank component (or said differently, with sub-optimal performance) until sufficient quantities of data are gathered. Finally, some research datasets do not provide the raw collections of documents but only standard features computed for each document from these collections, such as TF-IDF values. The problem with such datasets is that custom feature sets cannot be created. This is for instance the case with the LETOR datasets [79] that are essentially aimed at machine learning, or more precisely learning-to-rank, research.

Thankfully, a few corpora are still suitable for our purposes. In this work, we use the CSIRO Enterprise Research Collection (CERC) corpus [19]. The CERC corpus is a snapshot of all the public facing pages of CSIRO [1], Australia's national science agency, as of March 2007. The collection contains a total of 370,715 pages, which represents slightly more than 4GB of raw textual data. Sets of queries with gold-standard relevance judgements were created for the enterprise search tracks of the TREC competitions in 2007 [20] and 2008 [27].

More specifically, for the TREC 2007 competition, CSIRO employees were asked to create overview pages for 50 "topics", each listing a set of key documents [20]. In the TREC 2008 competition, 77 information requests from the public were used to define topics [27]. Then, from each topic was derived a query, the associated key documents being used as gold-standard judgements. Relevance judgements were on a three-point scale:

- 0 indicates that the document is not likely to be relevant,
- 1 that the document is possibly relevant, and
- 2 that the document is very likely to be relevant (a "key page").

Examples of queries include "high protein diet", "H5NI avian influenza" and "recruitment".

In all experiments, the number of search results per query is limited to 1,000 documents as is common practice, so longer lists of results are truncated.

4.4.2 Initial ranking evaluation

Effect of term weighting formulae

First, let us start by comparing the performance of our proposed term weighting formula based on meaningfulness scores with two state-of-the-art term weighting mechanisms, TF-IDF and BM25. The results are presented in Table 4.5.

On the TREC 2007 dataset, the best formula is BM25 according to most metrics, closely followed by our proposed term weighting. TF-IDF results in significantly worse performance on all metrics. However, the results are more even on the TREC 2008 dataset. TF-IDF now yields the best MRR and Mean $nDCG$@10, whereas BM25 comes first with the MAP and Mean $nDCG$ metrics. Our formula outperforms the other two on MAP@10.

Overall, our proposed term weighting formula seems highly competitive with the state-of-the-art formulae. It is very interesting to note that it consistently performs well (first or close second) on metrics that emphasize top results, that is MAP@10, MRR and Mean $nDCG$@10. This is evidently a desired behaviour as users expect the most relevant results to be returned first; in practice, users even prefer reformulating a query rather than looking at the tail of search results [125]. The reason why our scheme yields slightly worse MAP results might be due to the fact that many documents in the tail of search results will have 0 relevance scores (i.e., $score_{meaning}(D, Q)$ values), for example if the query contains a stop-word. In the current implementation, these

Table 4.5: Comparison of term weighting formulae for initial ranking. Higher values indicate better performance.

	Our weighting	TF-IDF	BM25
TREC 2007:			
MAP	0.316	0.216	**0.372**
MAP@10	0.670	0.242	**0.682**
MRR	0.896	0.451	**0.916**
Mean nDCG	0.778	0.628	**0.795**
Mean nDCG@10	**0.880**	0.494	0.864
TREC 2008:			
MAP	0.096	0.093	**0.146**
MAP@10	**0.176**	0.169	0.131
MRR	0.290	**0.358**	0.249
Mean nDCG	0.574	0.563	**0.611**
Mean nDCG@10	0.429	**0.451**	0.395

documents are randomly ordered. Therefore, if any relevant document contains only non-meaningful query terms, it will have a 0 score and might end up at a very low rank amongst many non-relevant documents. Furthermore, since search results are limited to 1,000 documents, some of these relevant documents might even be discarded. This issue can be alleviated in a simple way, by further sorting search results that have the same $score_{meaning}(D, Q)$ value by decreasing number of unique query terms it contains. Note that Mean $nDCG$ numbers, which also takes into account the full results list but gives a more direct emphasis to the top results than MAP, are particularly competitive.

Overall, these results are especially impressive when put in context. Indeed, both TF-IDF (or at least the popular LNC-LTC variant that we use) and BM25 are complex, finely-tuned formulae, that are the outcomes of decades of research. Nonetheless, our formula based on meaningfulness scores showed its effectiveness on the TREC datasets and proved to be on a par with state-of-the-art schemes. Hence, we shall continue building on it in the remaining of this chapter.

It should be pointed out that all three schemes have similar computational requirements as their respective formulae rely on similar quantities, that is frequencies relative to individual documents and frequencies relative to the full collection of documents. And in any case, these quantities are pre-computed during the indexing stage, which is much less time critical than the retrieval stage (i.e., query time).

Finally, our formula enjoys a much more sound theoretical grounding than TF-IDF based formulae. Also, because it relies on a simple model (see Section 3.2.1 page 18), its derivation is more straightforward than BM25's derivation and does not require many simplifying assumptions in order to make it computationally tractable [113].

Effect of index compression

Meaningfulness scores provide a natural distinction between important and non-important terms. Indeed, only terms with a strictly positive meaningfulness score correspond to keywords, i.e. words with an unusually large number of occurrences inside a document. Therefore, it is natural to wonder what the effect of the non-important terms on retrieval performance is. Are they needed at all?

To answer these questions, we modified our initial ranking formula to exclude all non-meaningful words. That is to say, only documents with a $score_{meaning}(D, Q) > 0$ are now considered relevant and returned as search results. Thus, part of the tail of results is discarded while the top of the results list (all results with at least one meaningful query term) is left untouched. But this simple modification has important practical consequences. Indeed, it can be seen as index compression: the inverted index now becomes much lighter because non-meaningful words no longer need to be indexed (i.e., stored). Also, since fewer entries need to be read from the index and processed at query time, query processing times are shortened. The impact of this experiment on the evaluation metrics are presented in Table 4.6.

For both TREC datasets, discarding non-meaningful words actually has very little impact on the numerical results. This confirms that most relevant documents contain meaningful query terms and that meaningfulness scores are an efficient approximation of relevance. It is important to point out that, despite a good performance on the TREC datasets, removing non-meaningful terms will decrease recall in most cases. This can have a detrimental impact on the quality of search results: indeed, some potentially relevant documents will no longer be returned by the search engine because all query terms inside these documents will be non-meaningful. This may be the case, for example, when only a small part of a document is relevant for the query. If the rest of the document discusses other topics, the frequencies of query terms are likely not to be high at the scale of the whole document.

On the positive side, we computed the number of meaningful terms in the entire collection in order to quantify the gains in terms of index storage space: 2.4 million terms are meaningful and 32.3 million are not. That is, if non-meaningful terms are

Table 4.6: Impact of filtering non-meaningful words on initial rankings.

	Default: all words	Meaningful words only
TREC 2007:		
MAP	**0.316**	0.306
MAP@10	**0.670**	0.660
MRR	**0.896**	**0.896**
Mean nDCG	0.778	**0.788**
Mean nDCG@10	0.880	**0.882**
TREC 2008:		
MAP	**0.096**	0.086
MAP@10	**0.176**	**0.176**
MRR	**0.290**	**0.290**
Mean nDCG	**0.574**	0.568
Mean nDCG@10	**0.429**	**0.429**

not indexed, 93% of the inverted index entries disappear, representing a significant reduction in storage space. Said differently, only 7% of unique words inside a document would be indexed on average. Note that the vocabulary size remains the same as all words can be indexed: there are still a few index entries for the word "the". These could be, for example, documents about determiners. To put these savings in perspective, stop-words filtering is estimated to reduce the number of index entries by 25 to 30% (see Section 5.1 in [86]).

Also, the two experiments carried so far (with and without non-meaningful terms) represent the two ends of a spectrum. Hybrid strategies can be devised: for example, one could discard non-meaningful terms that are in a pre-defined list of stop-words but leave all other terms untouched. This would still reduce the index size and speed up computations, while having less impact on recall.

In a nutshell, index compression may have detrimental effects on search results but these were shown to be negligible on the two TREC datasets. While it would be prudent for a standard search engine not to rely on this technique, it may prove a convenient and very powerful technique when storage space and/or computational resources are limited, e.g., with embedded systems.

4.4.3 Re-ranking top results with learning-to-rank

Now that we described several ways to efficiently compute an initial ranking of relevant documents in response to a query, let us investigate ways to further improve the rankings before showing them to the user, with the help of learning-to-rank. In all following experiments, the initial rankings are computed with our default term weighting formula, $score_{meaning}(D, Q)$, introduced in Equation (4.8).

As a quick reminder, a set of top documents is computed using our initial term weighting mechanism based on meaningfulness scores. For each such top document, a feature vector that captures many different aspects of relevance is then computed. These feature vectors then allow a learning-to-rank algorithm to permute these top documents to ensure the most relevant documents are placed at the top of the results list.

Learning-to-rank algorithms, as all supervised learning algorithms, need data to be trained on. To this end, queries in each TREC datasets are randomly divided into two sets. The first set, containing 80% of the queries, is used for training the learning-to-rank algorithms. The remaining 20% of queries is held out and used as a test set to evaluate algorithm performance: all numerical results reported below are reported on the test sets. The main reason for separating training data from test data is that the latter provides better generalization estimates. Indeed supervised learning algorithms may overfit, that is model the noise rather than the true signal from the data, in which case performance estimates computed from training data are unrealistically high.

Also, in the experiments below, the hyper-parameters of all learning algorithms are optimized using grid search and cross-validation.

Effect of learning algorithm on re-ranking

As already discussed, many machine learning algorithms can be used for re-ranking. Standard binary classification algorithms can be used in a pairwise fashion. Alternatively, listwise algorithms, specialized for re-ranking, can be used to directly minimize a listwise loss function such as MAP or $nDCG$ and as such they directly treat lists as training instances.

In this section, the performance of several re-ranking algorithms is studied and the results are presented in Table 4.7. Note that the numerical results before re-ranking are slightly different from the ones presented on Table 4.5; this is because the numbers presented here are computed using the test set only, which is a subset of the full dataset.

Two pairwise algorithms are tested: an SVM with a Gaussian kernel [40] and AdaBoost [51]. These two classification algorithms were chosen for their ability to learn complex decision boundaries. A listwise adaptation of AdaBoost called AdaRank [138] is also tried, optimizing for $nDCG@10$. In this experiment, the number of top documents to re-rank is set to 25. The reader should recall that the remaining tail of results, that is documents at ranks 26 and below, is not modified by the re-ranking procedure and is still returned as part of the final results.

Table 4.7: Impact of the learning-to-rank algorithm on re-ranking, with top 25 documents re-ranked.

	Before re-ranking	Gaussian SVM	AdaBoost	AdaRank
TREC 2007:				
MAP	0.281	0.279	**0.282**	0.267
MAP@10	0.538	0.526	**0.582**	0.432
MRR	0.789	0.855	**0.905**	0.648
nDCG	0.727	0.717	**0.731**	0.683
nDCG@10	0.786	0.721	**0.797**	0.601
TREC 2008:				
MAP	0.107	**0.112**	0.111	0.108
MAP@10	0.198	**0.546**	0.510	0.288
MRR	0.304	**0.739**	0.651	0.548
nDCG	0.673	**0.705**	**0.705**	0.681
nDCG@10	0.510	**0.739**	0.730	0.548

On the TREC 2007 dataset, all metrics agree to designate AdaBoost as the best re-ranking algorithm. The SVM with a Gaussian kernel improves MRR but slightly decreases all other metrics, compared to the initial rankings. On the 2008 dataset, SVM now performs best, with AdaBoost being close second: both algorithms now bring large improvements compared to the initial results before re-ranking. Rather surprisingly, the listwise algorithm AdaRank does not perform well: it significantly degrades the initial results on the 2007 dataset and brings only small improvements on the 2008 dataset. Investigating, we found that the algorithm severely overfits on both datasets. The explanation for this behaviour is the limited number of queries available for training (recall that listwise algorithms treat one query as one data point).

Comparing the two pairwise algorithms, AdaBoost seems to be slightly superior especially on Mean $nDCG@10$ and is the algorithm chosen moving forward. AdaBoost was also roughly 10 times faster than SVM (11ms per query, as opposed to 140ms) in our experiments, which further strengthens our preference for AdaBoost. However, it is difficult to say which algorithm will be faster in general because the two algorithms are not directly comparable. AdaBoost is a boosting algorithm, which output relies on a sequence of weak classifiers, typically decision trees. The length of this sequence can range from a dozen to several hundred weak classifiers. The prediction of each weak classifier is fast but the sequentiality of the problem makes computation parallelism impractical. SVM relies on a completely different mechanism, with prediction time proportional to the number of so-called support vectors. If the classifier does not overfit, this number should be a small percentage of the training dataset, which can still be large in absolute terms.

Effect of the number of top documents used for re-ranking

The number of 25 top documents used for re-ranking in the previous experiment was an educated guess. Let us now study more thoroughly the effect of the number of top documents on re-ranking performance. To do so, 10 models are trained, starting with 5 top documents and going up to 50, in increments of 5 documents.

The results with 10, 25 and 50 top documents are presented in Table 4.8. Figures 4.1a to 4.1d show the evolution of various metrics using all 10 models.

Analyzing the numbers in Table 4.8, using 10 top documents already has a large beneficial impact on the TREC 2008 results but it has a detrimental effect on TREC 2007. Increasing the number of top documents yields good improvements on both datasets and the best results are obtained with 50 top documents for almost all metrics.

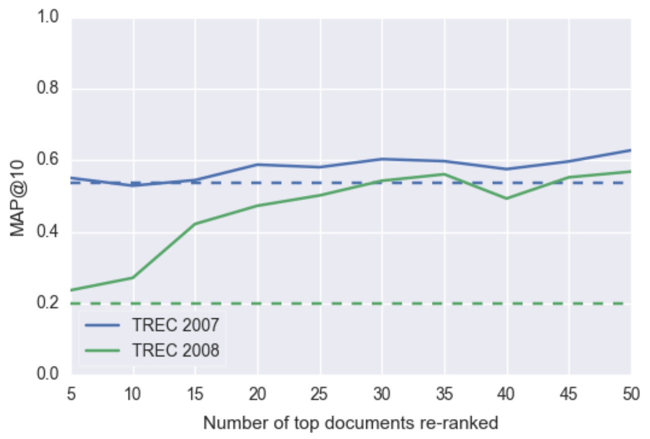
Looking at the plots confirms this analysis. Re-ranking gives good increases on all metrics starting from 15 to 20 top documents, the improvements on TREC 2008 being particularly impressive. Furthermore, the increases for all metrics seem to plateau almost immediately, despite some fluctuations. And as shown in Figure 4.1d, increasing the number of top documents comes with a certain computational cost: indeed, the re-ranking cost increases quadratically with the number of top documents. This is because the number of pairs of top documents is $\binom{n}{2}$, where n denotes the number of top documents. However, methods to speed up pairwise re-ranking have been proposed [65]; also, listwise approaches are usually more efficient as they do not rely on creating document pairs.

As such, our initial choice of 25 top documents seems very reasonable. However,

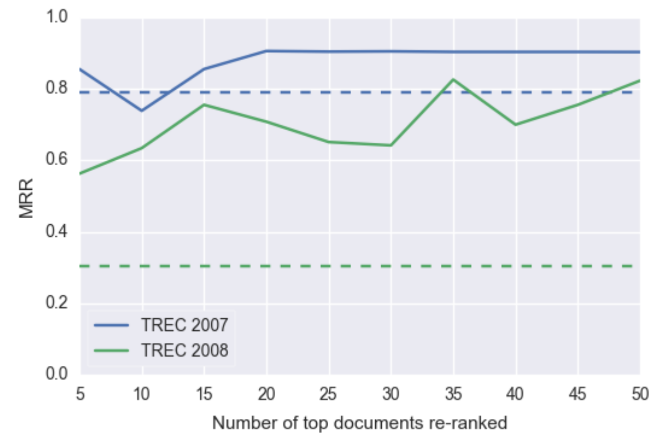
Table 4.8: Impact of the number of top documents on re-ranking, using a pairwise AdaBoost algorithm.

	Before re-ranking	10 documents	25 documents	50 documents
Trec 2007:				
MAP	0.281	0.280	0.282	**0.285**
MAP@10	0.538	0.529	0.582	**0.628**
MRR	0.789	0.739	**0.905**	0.903
nDCG	0.727	0.722	0.731	**0.732**
nDCG@10	0.786	0.779	0.797	**0.812**
Trec 2008:				
MAP	0.107	0.108	0.111	**0.113**
MAP@10	0.198	0.271	0.510	**0.569**
MRR	0.304	0.634	0.651	**0.823**
nDCG	0.673	0.685	0.705	**0.710**
nDCG@10	0.510	0.653	**0.730**	0.724

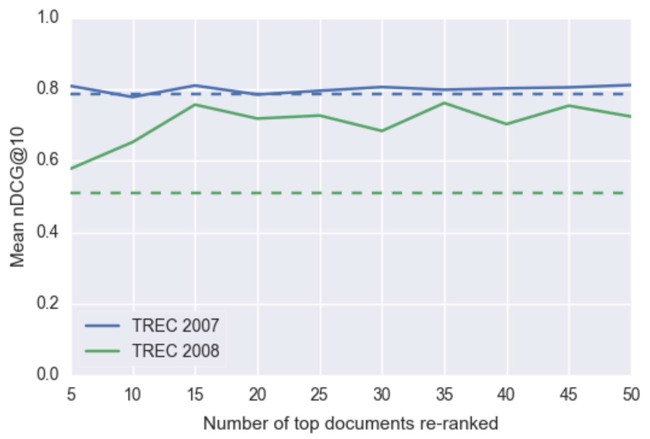
this number should be ideally tuned for each collection, depending on a number of factors, such as the number of documents and the redundancy of information in the collection. Redundancy of information, for example, should be low with enterprise documentation of good quality but it ineluctably increases over time with the launch of different projects, employee turnover, etc.



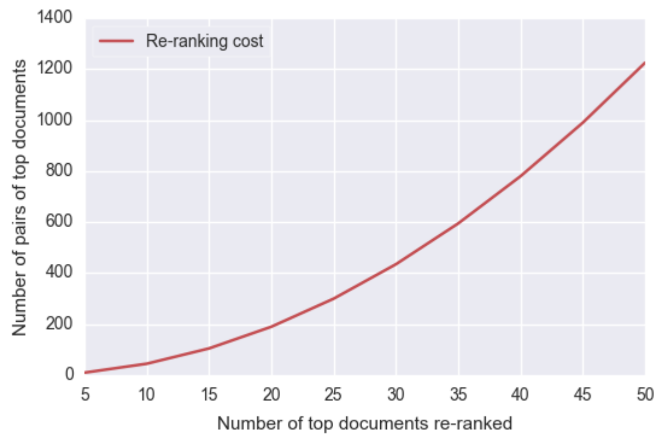
(a) Mean Average Precision at rank 10



(b) Mean Reciprocal Rank



(c) Mean nDCG at rank 10



(d) Computational cost of pairwise re-ranking

Figure 4.1: Impact of the number of top documents re-ranked on various metrics and on the computational cost. Dashed lines indicate baseline performance, that is, before re-ranking.

4.5 Conclusion

Search engines have become ubiquitous and are indispensable tools to deal with information overload. But the impact of information retrieval is not limited to search engines. The discipline has also had a large impact on text mining. Search is becoming a standard component of many systems but it also influenced the development of many related disciplines like information filtering and query-focused summarization, to name a few.

In this chapter, we proposed a new term weighting mechanism for information retrieval. A fully functional search engine, primarily aimed at enterprise search, has been built upon it. It includes re-ranking using a rich set of features and learning-to-rank algorithms.

We evaluated and tuned our search engine with the TREC Enterprise Search datasets, from 2007 and 2008. We found that, on these datasets, our term weighting mechanism is competitive with state-of-the-art schemes like TF-IDF and BM25. A small modification allowing to dramatically decrease the index size was also proposed and it was shown to have almost no detrimental impact on retrieval quality. We then turned our attention towards learning-to-rank. We found pairwise approaches to be effective and more appropriate than their listwise counterparts when few gold-standard queries are available for training purposes. However, the best binary classifier to use in a pairwise setting was dependent on the dataset. The effect of the number of top documents re-ranked has also been studied; it was found that using a number large enough is important (around 15-20 with our datasets) after which performance gains stagnate.

A fully-fledged search engine has been developed as part of this work and used by a large business unit inside HP. It contains features that are not described in this chapter, such as search result filters (to restrict results by dates, authors, etc.) and "related terms" suggestions to help users reformulate their queries. It also collects automatically both implicit and explicit user feedback to automatically create datasets of queries and allow the search results to improve over time. The software has been written to allow new features to be incorporated easily and new experiments to be conducted. Adding spelling correction to the query text and automatic query reformulation are obvious next steps for future developments.

Chapter 5

Text Segmentation

Automatically segmenting a text into consistent topical segments is a fundamental problem in text mining. Even a moderately long document may consist of several relatively independent topics and parts. Such heterogeneity in documents can seriously affect the performance of classification and other mining algorithms.

Let us assume one is trying to automatically detect the sensitivity level of a document for a government agency. In the United-Kingdom, there are three main classification policies: Official, Secret and Top-Secret [14]. The main difficulty in this setting comes from the fact that most documents are not monolithic blocs but rather contain a mixture of topics with different sensitivity levels and therefore different security restrictions. For example, a military report might contain a few paragraphs regarding atomic weapons secrets diluted with publicly known information. Standard automatic classification algorithms will tend to incorrectly classify the document as non-critical. A potential solution is to segment the document into homogeneous topical parts, categorize those parts individually and apply the most restrictive security policy to the entire document.

In this chapter, an algorithm for automatic text segmentation is presented. A brief review of the state-of-the-art is introduced in Section 5.1. The proposed algorithm is described in Section 5.2. The evaluation methodology is presented in Section 5.3 and the evaluation results in Section 5.4. Conclusions are drawn in Section 5.5.

A library implementing the proposed algorithm has been developed and a demonstration program has been built around it. A brief description of both of these is available in Appendix A.2, page 110.

5.1 A brief literature review

Text segmentation has been a large research area in the past twenty years and many different techniques have been proposed. The most popular and influential approaches are reviewed below.

The vast majority of text segmentation algorithms rely on the same underlying idea: the quantification of lexical cohesion between different parts of a document. Lexical cohesion can be defined as “the cohesive effect achieved by the selection of vocabulary” (see page 274 of [58]) which intuitively means that changes in the vocabulary signal topic changes. A number of linguistic structures create lexical cohesion, such as word repetitions, synonyms, pronouns and more generally related vocabulary. However, many of these structures can be difficult to detect due to language ambiguities and variabilities. This is why, even if more sophisticated alternatives exist [95], many popular and effective approaches to lexical cohesion quantification are solely based on some measure of word repetitions.

In one of the earliest algorithm *TextTiling* [61], Hearst uses two adjacent blocks of length 120 words (see Figure 5.1) and calculates the cosine similarity between these two blocks as:

$$sim(b_1, b_2) = \frac{\sum_w f(w, b_1) f(w, b_2)}{\sqrt{\sum_w f(w, b_1)^2}\sqrt{\sum_w f(w, b_2)^2}} \tag{5.1}$$

where $f(w, b)$ is the frequency of the word w in the block b. This similarity is attached to the gaps between the blocks; here, a gap corresponds to a space between two consecutive words.

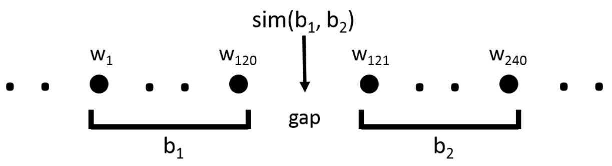
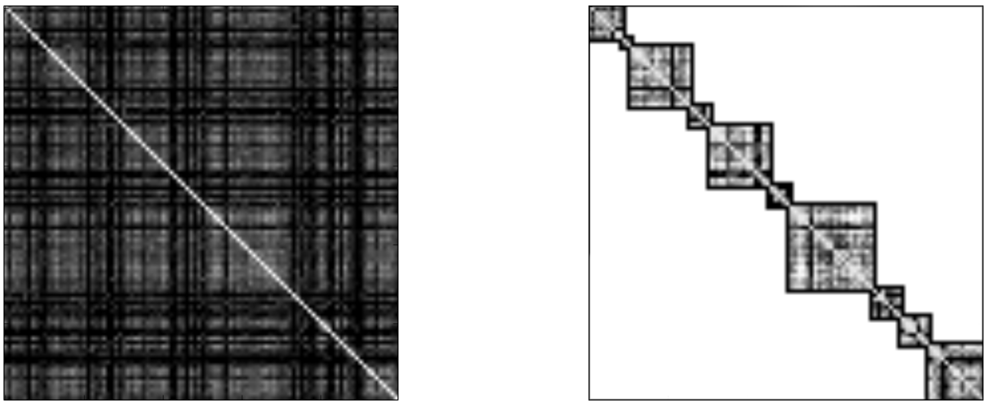


Figure 5.1: Cosine similarity computation between adjacent blocks of 120 words, in the TextTiling algorithm [61]. w_i**s represent words.**

By moving the two blocks along the document (shifting the blocks to the right by 20 words at a time), we can graphically represent the calculated cosine similarity as a function of gaps. The author then introduces a depth score, defined as the average of the relative depths between the gap value and the two closest local maxima, one on the left and one on the right. The gaps with the biggest depth scores are selected using some automatic threshold and these gaps are used to divide the document into segments. Let us point out that these gaps are between words and not between sentences, so small additional adjustments need to be done.

Choi applies methods and ideas from image processing in his algorithm *C99* [36]. If a text document consists of n sentences, then an $n \times n$ gray image is constructed with "pixel" values at (i, j) equal to the similarity $sim(s_i, s_j)$ between sentences s_i and s_j (see Equation (5.1)). An example of such an image is presented in Figure 5.2a; light colors represent high similarities. As we can see from the picture, the resulting


(a) Cosine similarity matrix (before rank filtering)

(b) Patches extracted by the divisive clustering procedure

Figure 5.2: Main steps of the C99 algorithm [36]

image has several light patches around the diagonal which correspond to groups of consecutive sentences with high cosine similarities between them. To extract such disjoint objects, the image of similarity scores is first filtered using an 11×11 rank mask. Then coherent square regions near the diagonal of the image (i.e., segments) are extracted using a divisive clustering procedure that maximizes their density, as illustrated in Figure 5.2b. As in TextTiling [61], the number of segments is determined by some empirical threshold.

A more rigorous approach to automatic segmentation, using a Bayesian statistical model, was developed by Utiyama and Isahara in [130]. Their algorithm *TextSeg* subsequently influenced many new developments and generalizations. Let us denote a text document by D and the set of all possible segmentations by $\mathcal{F}$. The optimal segmentation is defined as the segmentation $\hat{S} \in \mathcal{F}$ that has maximum probability given the document D, i.e.:

$$\hat{S} = \underset{S \in \mathcal{F}}{\operatorname{argmax}} P(S|D). \tag{5.2}$$

Using Bayes formula, we can rewrite Equation (5.2) as:

$$\hat{S} = \underset{S \in \mathcal{F}}{\operatorname{argmax}} P(D|S)P(S) \tag{5.3}$$

$$= \underset{S \in \mathcal{F}}{\operatorname{argmin}} \left[-\log \Big(P(D|S)P(S) \Big) \right]. \tag{5.4}$$

The authors in [130] impose strong independence assumptions and present the following models for the likelihood $P(D|S)$ and the prior $P(S)$:

$$P(D|S) = \prod_{i=1}^{m} \prod_{j=1}^{n_i} \frac{f(w_j, S_i) + 1}{n_i + V} \tag{5.5}$$

$$P(S) = n^{-m} \tag{5.6}$$

where:

- m is the number of segments in S,
- n is the number of words in D,
- n_i is the number of words in segment S_i,
- $f(w_j, S_i)$ is the frequency of the word w_j in S_i, and
- V is the vocabulary size (i.e., the number of unique words in D).

To find the optimal segmentation, the following complete graph G is constructed: the vertex set is the set of all gaps between sentences $\{g_i : i = 1, \ldots, n-1\}$ plus two additional vertices START and END corresponding respectively to the beginning and end of D, as illustrated in Figure 5.3. From Equations (5.4), (5.5) and (5.6), the weight c_i of the edge corresponding to the segment S_i, between the two vertices encompassing all words in the segment, is defined as:

$$c_i = \log n + \sum_{j=1}^{n_i} \log \frac{n_i + V}{f(w_j, S_i) + 1}. \tag{5.7}$$

Then, the solution of Equation (5.2) corresponds to the shortest path between START and END in the weighted graph G, which can be found efficiently using dynamic programming.

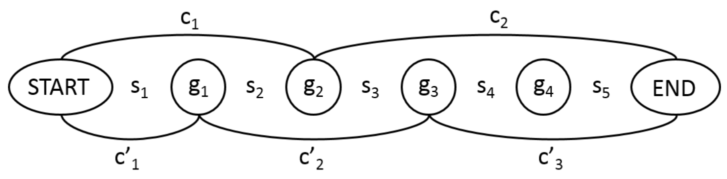


Figure 5.3: The graph G for an example document of 5 sentences in *TextSeg* [130]; for clarity, some edges are omitted. Two paths between START and END, corresponding to two possible segmentations, are shown. The g_i vertices correspond to gaps between the sentences s_i and s_{i+1}.

Eisenstein and Barzilay [48] introduced *BayesSeg*, an extension of *TextSeg* [130], in which they developed a more general model for $P(D|S)$ based on Latent Dirichlet allocation (LDA) [31]. LDA is a very natural approach to text segmentation since we can think of segments as types of topics inside a document. This makes *TextSeg* more general and flexible at the expense of a more difficult analysis.

A few other influential approaches are worth mentioning. Galley et al.'s *LCSeg* algorithm [52] weights lexical chains with a variant of TF-IDF and then computes a cosine similarity score for each gap, based on the weights of chains overlapping two adjacent moving windows. Malioutov and Barzilay [84] build a fully connected graph and use a Minimum Cut algorithm to find the optimal segmentation. The edge weights are pairwise sentence cosine similarities, similarly to C99 [36], but also take into account neighboring sentences to get smoother similarities. Finally, Yamron et al. [139] compare text segmentation to speech segmentation, a necessary step in speech recognition. Indeed, segmenting a text document into topics is in some respects similar to segmenting a sound signal into a sequence of phonemes. Therefore, speech segmentation algorithms can be adapted to text segmentation and the authors develop a topic modeling approach based on Hidden Markov models. However, contrary to the approaches introduced so far, this algorithm requires a training corpus to compute the parameters for the topic models.

Even though this is not the focus of this chapter, one should also add for completeness that supervised learning techniques are often used to improve the algorithms' performance in practical systems. Those techniques also allow to incorporate domain knowledge and specialize the system for the task at hand: for example, cue phrases, silences and speaker changes are effective features for speech segmentation [78, 52].

5.2 Proposed algorithm

In this section, we present a new approach to automatic text segmentation based on the meaningfulness measure introduced in Equation (3.11), page 22.

We will assume in our work, without loss of generality, that segment boundaries need to be placed at the location of gaps (i.e., breaks) between consecutive sentences. Depending on the application, one could need to place boundaries between clauses, sentences, paragraphs or even changes of speaker when working with human conversations. Our algorithm can be trivially extended to all these cases, by choosing another set of gaps.

Inspired by previous work, the main idea of our algorithm is to calculate, for each gap in the document to segment, a measure of lexical cohesion between the sentences around that gap. The measure of meaningfulness, derived from the Helmholtz principle in Section 3.2, is computed with the help of sliding windows and used to locally quantify lexical cohesion. After that, the gaps where the biggest drops in cohesion occur will be selected to segment the document.

5.2.1 Preprocessing

Let D denote a text document. If necessary, D is first split into sentences by using a regex-based tokenizer considering the characters ".", "!" and "?" as sentence delimiters. Then each sentence is split into a list of words, using all non-alphabetical characters as delimiters and all words are downcased. Stop-words removal is optional and it is actually most of the time detrimental as will be observed later during the evaluation section. Finally, the Porter stemming algorithm is applied and only stems of length at least two are considered. After this preprocessing the document D is the sequence of n sentences $s_1, s_2, \dots, s_n$.

5.2.2 Gap scores calculation

Text documents typically contain several weakly related or unrelated topics, or parts. Our goal is to identify topically coherent segments in a document D with n sentences. From the $n-1$ gaps between the document sentences, we are looking for the gaps which best separate the different topics inside the document (see Figure 5.4).

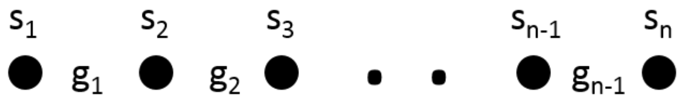


Figure 5.4: Segmenting is finding which of the $n-1$ gaps between sentences best separate the topics inside the documents.

In most documents topics have a hierarchical structure. Indeed, a book is divided into chapters, each chapter being divided into sections, which are themselves divided into paragraphs. The desired segment size will depend on the applications and requirements, so we believe a generic segmentation algorithm should not prefer one level of segmentation over another. This is why our algorithm takes a single input parameter: the number of segments desired. We denote this parameter by m.

For completeness, it should be pointed out that when the exact number of segments m is unknown, one could run the algorithm with different values for m and choose the one that minimizes a global metric to be defined. Alternatively, the algorithm could be modified to refine the number of segments automatically. The difficulties of such approaches are briefly discussed during evaluation.

Let us now present the main steps of the proposed algorithm for text segmentation. Our algorithm works by quantifying lexical cohesion between several consecutive sentences using *gap scores* that are attached to each one of the $n-1$ gaps. These gap scores are computed using sliding windows and the weighted measure of meaningfulness $meaning$ from Equation (3.11), page 22, as follows.

The $n-1$ gap scores are initialized to 0. The window size is set to the average segment size desired, which is simply computed as $round(n/m)$ since the desired number of segments, m, is given.

Then, for each window, the weighted meaningfulness scores of all words inside the window are computed; words with a *strictly positive* score are declared meaningful inside that window. It is important to note that some meaningful words may be present only in a small portion of the window; therefore, such a word should only have an influence in the part of the window where it appears. We call this "part" the *activity stretch* of the word inside the window. Formally, the activity stretch of a word w inside a window W_i is the group of consecutive sentences between the *first* and *last* occurrence of w in the sentences inside W_i, as illustrated in Figure 5.5. For each meaningful word in each window, the gap scores corresponding to gaps in the current window are updated by adding the meaningfulness score to all gap scores encompassed by the activity stretch of the current word. Thus, larger gap scores indicate stronger lexical cohesion between the sentences before and after the corresponding gaps.

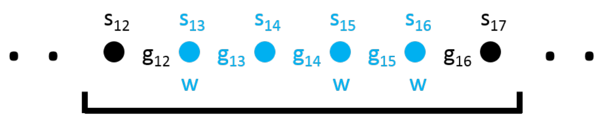


Figure 5.5: Activity stretch: the word w, meaningful in the current sliding window, contributes only to the gap scores in the part of the window where it appears. Here w occurs in sentences 13, 15 and 16, so its activity stretch is (s_{13}, s_{16}); thus it contributes to the gap scores of gaps g_{13}, g_{14} and g_{15}.

Note that since there are more windows in the middle of a document than in its beginning or end, we use periodic boundary conditions for defining sliding windows, assuming the document sentences are arranged in a circular fashion. The exact algorithm for gap scores calculations is summarized in Algorithm 5.1.

Algorithm 5.1: Gap scores calculation.

```
Initialize all gap scores to 0.
For all possible sliding windows $W_i$ (using periodic boundary
conditions) :
    For each word $w$ appearing in $W_i$ :
        Compute $meaning(w, W_i, D)$ [see Equation (3.11) page 22].
        If $meaning(w, W_i, D) > 0$ :
            ⋆ Determine the activity stretch of $w$ inside $W_i$.
            ⋆ Add $meaning(w, W_i, D)$ to the gap score of each gap
            encompassed by the computed activity stretch.
```

This moving windows approach allows us to compare the local word frequencies, in each window, to the frequencies in the whole document. Also, combining a large window size (set to the expected segment size) together with words' activity stretches allows us to capture the lexical cohesion created by both short-range and long-range word repetitions (i.e., from words repeated very locally in a few sentences to word repetitions spanning entire segments).

Intuitively, a small gap score indicates that a small number of meaningful words have their activity stretch encompassing the associated gap and/or that those words are less meaningful (i.e., have smaller weighted scores) than in other parts of the document. Thus, a small gap score indicates low lexical cohesion between sentences before and after the gap, making it a good candidate for placing a segment boundary.

Figure 5.6 shows the typical shape of the gap scores curve for test documents. Sharp drops in gap scores can be observed at the location of ground truth cuts. The simplest approach to identify the best cuts (i.e., segment boundaries) is to look for the smallest gap scores. Unfortunately, as we can see from the figure, the gap scores function is typically very irregular and noisy. Indeed, it is a highly variable function with a lot of local minima, even in the vicinity of the global minimum. So, to identify gaps with sharp drops, some regularization of the gap scores function is required.

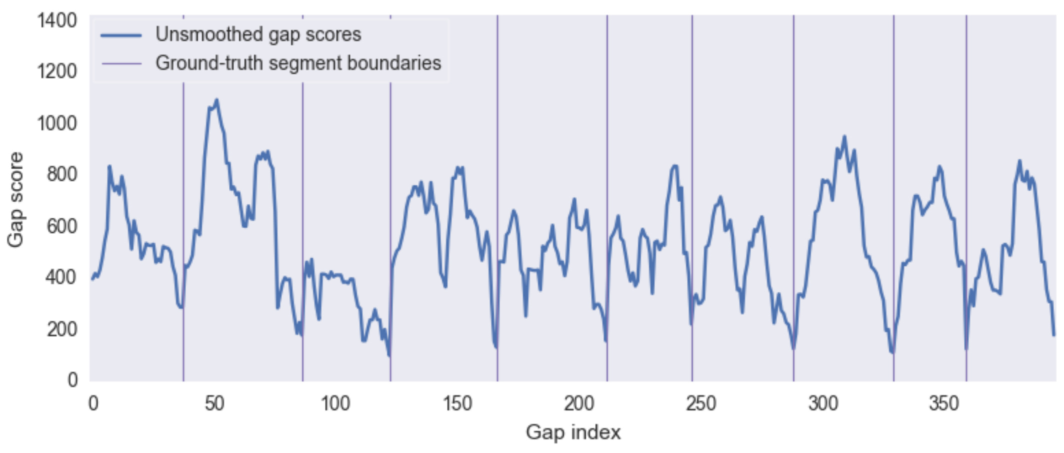


Figure 5.6: Unsmoothed gap scores and ground truth boundaries. The ground truth boundaries coincide, as expected, with sharp drops in gap scores.

5.2.3 Identification of segment boundaries

To regularize the gap scores function we shall follow the well-known approach from image processing to similar problems. We use the *diffusion equation* (or the one-dimensional *scale-space* theory):

$$u_t = \frac{1}{2} u_{xx} \tag{5.8}$$

to smooth the gap score curve (i.e., $u|_{t=0}$ is the gap scores function), and find stable and reliable critical points. In our case, the crucial property of the one-dimensional scale-space theory is that *increasing the smoothing parameter (or diffusion time) results in the decrease of the number of critical points* (see [77] for details). Note that this property is valid only in one dimension and not in any other dimensions.

From the numerical point of view, the evolution of the gap scores function under the diffusion equation is just a convolution of this function with a Gaussian filter:

$$G(x;\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{x^2/2\sigma^2}, \tag{5.9}$$

where $\sigma^2 = t$. Using the three-sigma rule (i.e., so that 0.9973 probability under the normal distribution is concentrated in a neighborhood of size 3σ around the mean value), we convolve the filter only in a neighborhood of size 3σ around every gap. Figure 5.7 illustrates gap scores smoothing with different values for the parameter σ. Since increasing the smoothing parameter decreases the number of local minima, our strategy is to increase σ with small steps until the number of minima reaches the desired number of cuts. Alternatively, for faster computations, one can start from a large value of σ giving too few local minima and refine it using dichotomy.

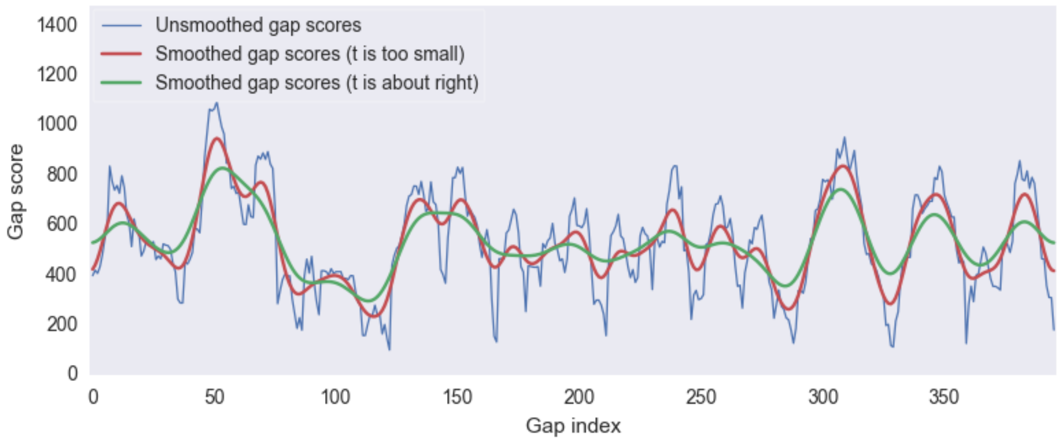


Figure 5.7: Effect of the parameter σ for smoothing the gap score curve. σ is gradually increased until the desired number of local minima is obtained.

An undesirable effect of this diffusion process is that each point shifts, on average, by a distance less than or equal to σ. This is a well-known property of the Brownian motion. This phenomenon can be observed in Figure 5.8. We are only interested in the locations of the critical points but one cannot predict the exact shift of any given point. Some points will shift by a small distance only from their original location, while some others will go much farther (i.e., at a distance greater than σ from their original location).

In our algorithm, we consider that a critical point can shift at a distance up to 1.5σ. So, we first calculate the shifted critical points (i.e., the local minima computed from the smoothed curve). After that we find the "true" local minima in the *original* noisy

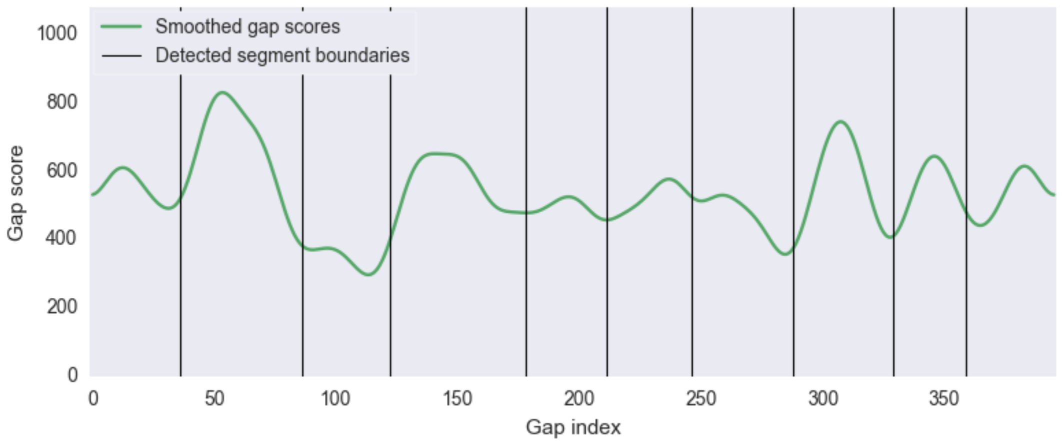


Figure 5.8: Smoothed gap scores and final boundaries identified by our algorithm. Those boundaries are placed in local minima from the *unsmoothed* signal. We can clearly see that most boundaries are slightly away from the *smooth* local minima.

curve in the 1.5σ vicinity of the shifted critical points. The segment boundaries returned by our automatic text segmentation algorithm are those "true" local minima.

Figure 5.9 shows a comparison of the algorithm output with the ground truth segmentation of our example document.

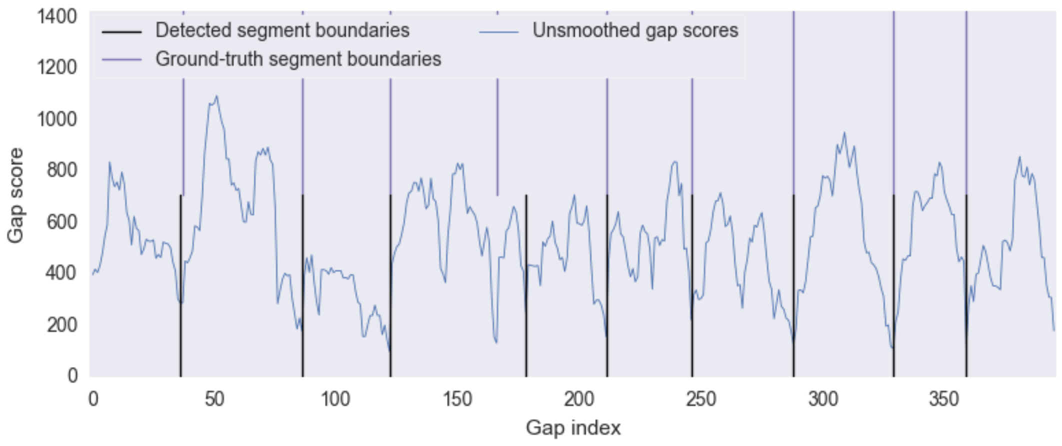


Figure 5.9: Comparison of the boundaries from the ground truth segmentation and from the output of our algorithm. Most boundaries are identified precisely, except the first and fourth ones that are off by 1 and 12 sentences respectively.

5.3 Evaluation methodology

5.3.1 Evaluation procedure

We compare our algorithm to two state-of-the-art algorithms: *C99* [36] and *BayesSeg* [48]. We use the public domain implementations released by their respective authors. Two baseline algorithms are also used. The *Uniform* algorithm creates segments of equal size, while *Random* places segment boundaries randomly.

In all experiments, we consider the number of segments in each document known and given. This is a common assumption used for evaluation and comparison; the reasons are two-fold.

First, topics have inherently a hierarchical nature and segmentation can usually be done at different scales. Indeed, a long segment that discusses a broad topic is likely to discuss several sub-topics. For example, the chapter dedicated to the standard library of a C++ book may describe, in turn, basic containers, strings, streams and numeric tools. Therefore, the scale of segmentation (or, similarly, the number of segments desired) should be an input of the segmentation algorithm.

Second, inter-agreement between human annotators is generally very low. Even when receiving similar instructions, annotators tend to select widely different segment sizes on the same data. In [84], four annotators were asked to manually segment physics lecture transcripts; the average number of segments created by the different annotators ranged between 6.6 and 18.4. Naturally, algorithms face the same problem. This issue is even exacerbated as automatic procedures for determining the number of segments in a document are likely to be domain dependent, making comparisons difficult. As an illustration, we experimented with the *TextTiling* algorithm [61] (Choi's implementation with default parameters) and the "3-5" subset of Choi's dataset [36] (which will be described in more details in Section 5.4.1). The average number of segments produced by *TextTiling* is 23, when each document actually contains 10 segments. Inevitably *TextTiling* would score poorly; however, such a poor performance reflects an inadequate automatic selection of the number of segments but does not allow to fairly judge the intrinsic qualities of *TextTiling*.

We will also experiment with the effect of stop-words filtering on the results and, to this end, we modified both *C99* and *BayesSeg* implementations to make stop-words filtering optional. Indeed, stop-words removal is an extremely common preprocessing step across all natural language processing techniques and algorithms, but it does create undesirable effects. Furthermore, there is not any universal list of stop-words. In the case of text segmentation, we argue that stop-words may contribute to the creation of lexical cohesion in a document and that, in most cases, stop-words can be helpful features for segmentation. Documents where each part has been written by a different author are clear examples; such documents include streams of news stories, specialized books where each chapter is written by a different expert, documents written collaboratively, etc. It is then reasonable to assume that each author will have his own writing style and will use a slightly different set of stop-words. Filtering them inevitably means discarding useful information.

5.3.2 Evaluation metrics

Let us explain the general approach to evaluating the performance of text segmentation algorithms.

For each document from the evaluation dataset, we need to estimate how different the reference segmentation is from a candidate segmentation obtained with an automatic segmentation algorithm. Using a rigorous mathematical approach, one would proceed as follows. If a document D is a sequence of sentences $s_1, s_2, \ldots, s_n$, then any segmentation is a subset of the space of gaps: $Gaps = \{g_1, g_2, \ldots, g_{n-1}\}$ (see Figure 5.4).

This space of gaps can be turned into a metric space by introducing a distance $d(g_i, g_j)$ between any pair of gaps g_i and g_j. A simple choice for $d(g_i, g_j)$ is $|j - i|$, i.e., the number of sentences between the two gaps. Alternatively, if we also want to take into account the number of words in the sentences, $d(g_i, g_j)$ can be defined as the sum of the sentence lengths (in number of words) between the two gaps. To take into account the document length, this distance can be normalized by scaling the diameter of gaps to one.

Now, the segmentations S and S' are subsets of the metric space $Gaps$. Therefore, the Hausdorff distance between subsets [4] can be used to evaluate the distance between the two segmentations. Unfortunately, the Hausdorff distance is very sensitive to outliers so some averaging is needed.

Currently, the most popular distance between segmentations is the WindowDiff distance (WD) [104]. It computes the distance between a reference segmentation R

and a candidate segmentation C as follows:

$$WD(R,C) = \frac{1}{n-k}\sum_{i=1}^{n-k} \mathbb{1}\left[B_R(i,i+k) \neq B_C(i,i+k)\right] \tag{5.10}$$

where:

- n is the number of sentences in the document,
- k is the window size, in number of sentences, set to half the average segment size in the reference segmentation,
- $B_S(i,j)$ is the number of boundaries between sentences s_i and s_j in the segmentation S, and
- $\mathbb{1}$ is an indicator function:

$$\mathbb{1}[c] = \begin{cases} 1 & \text{if c is true,} \\ 0 & \text{otherwise.} \end{cases} \tag{5.11}$$

In words, the WindowDiff distance works by moving a window of size k along the document. It adds a penalty of 1 for each window where the number of segment boundaries differs between the reference and candidate segmentations. The sum of penalties is then divided by the total number of windows to get a normalized score with values between 0 and 1.

Lamprier et al., in [72], noticed that errors at the beginning and end of a document are less penalized than errors in the middle of the document. This is corrected by adding $k-1$ fictitious sentences at both ends of the document.

For evaluation, we shall be using the WindowDiff distance, with the Lamprier et al.'s correction [72].

5.4 Experimental results

5.4.1 Synthetic datasets

Evaluating and comparing segmentation algorithms can be tricky as it is often difficult to define what the reference (i.e., ground-truth) segmentation of a document should be. This is why most evaluation datasets are synthetic and contain documents created by concatenating segments of varying sizes and from different sources.

We use Choi's synthetic dataset [36] which is commonly used for evaluation and contains 700 English language documents, each made of 10 segments[1]. Each segment consists of the beginning of a randomly sampled document from the Brown corpus. The segment sizes are chosen randomly and range between 3 and 11 sentences. The exact composition of the dataset is given in Table 5.1; the reader may refer to [36] for more details on this corpus.

Table 5.1: Composition of Choi's synthetic dataset [36].

Segment sizes, in number of sentences	Number of documents
3-5	100
3-11	400
6-8	100
9-11	100

Segments with lengths ranging between 3 and 11 sentences are quite short and practical problems may require to deal with much longer segment sizes and much larger volumes of data. For this reason, we created another dataset containing 100 documents[2]. Each document is again made of 10 segments, with segment sizes ranging between 30 to 50 sentences. We will refer to this dataset as "30-50". Similarly to Choi's methodology, we use articles sampled from the *News* and *Learned* categories of the Brown corpus.

Table 5.2 shows the evaluation results when stop-words are removed during preprocessing. The default list of stop-words for each algorithm is used. Choi's implementation of *C99* uses a list of 327 words while Eisenstein's implementation of *BayesSeg* uses 329 stop-words. In our algorithm, we use the default list provided by the Natural Language ToolKit library (NLTK) [6], consisting of 127 words. In this setting, *BayesSeg* consistently outperforms both *C99* and our algorithm, in most cases by a relatively wide margin. On short documents, *C99* performs slightly better than our algorithm while we outperform *C99* on longer documents.

However, all three algorithms behave quite differently when stop-words are not removed, as shown in Table 5.3. Our algorithm now gives better results than both *C99*

[1]These documents can be found in the folders *data/{1,2,3}/* in the package made publicly available by the author.

[2]This new dataset can be downloaded from the author's webpage.

Table 5.2: WindowDiff scores (using Lamprier et al.'s correction) on the synthetic datasets *with* stop-words removal. As a reminder, smaller scores are better.

	C99	BayesSeg	Our algorithm	Uniform	Random
Choi 3-5	0.129	**0.106**	0.168	0.380	0.532
Choi 3-11	0.140	**0.108**	0.191	0.456	0.531
Choi 6-8	0.110	**0.069**	0.160	0.223	0.548
Choi 9-11	0.093	**0.057**	0.128	0.189	0.537
30-50	0.078	**0.014**	0.050	0.274	0.522

and *BayesSeg*. The performance of *C99* is much worse in this case. The most likely explanation is that the pairwise sentence similarities are now completely dominated by insignificant words. The performance of *BayesSeg* is also degraded in this setting. On the contrary, our algorithm performs in most cases best when stop-words are not filtered.

Table 5.3: WindowDiff scores (using Lamprier et al.'s correction) on the synthetic datasets *without* stop-words removal.

	C99	BayesSeg	Our algorithm	Uniform	Random
Choi 3-5	0.340	0.189	**0.141**	0.380	0.532
Choi 3-11	0.378	0.182	**0.176**	0.456	0.531
Choi 6-8	0.356	0.178	**0.143**	0.223	0.548
Choi 9-11	0.385	0.141	**0.109**	0.189	0.537
30-50	0.466	0.074	**0.052**	0.274	0.522

Table 5.4 shows the influence of the stop-words list on the results. Each algorithm was tested using two different lists: the first one is a list of 127 words (the default for our algorithm, from NLTK) and the second one a common list of 571 words [5]. As can be seen from the table, the stop-words list chosen can have a significant impact on the performance of the algorithm. Furthermore, apart from *C99* where larger lists are preferable for reasons already explained, the list giving the best results is dependent on the dataset.

Note that, regarding computational performance, all three algorithms have similar running times on short documents so exact numbers are not presented in this section. Differences in computational requirements will be discussed in depth in the next section, when they become much more apparent.

Table 5.4: WindowDiff scores (using Lamprier et al.'s correction) on the synthetic datasets with different stop-words list. In each cell, the first score is obtained using a list of 127 words and the second using a list of 571 words.

	C99	BayesSeg	Our algorithm	Uniform	Random
Choi 3-5	0.138 / 0.122	**0.109 / 0.110**	0.168 / 0.180	0.380	0.532
Choi 3-11	0.146 / 0.144	**0.113 / 0.108**	0.191 / 0.195	0.456	0.531
Choi 6-8	0.117 / 0.105	**0.072 / 0.078**	0.160 / 0.176	0.223	0.548
Choi 9-11	0.097 / 0.094	**0.054 / 0.057**	0.128 / 0.127	0.189	0.537
30-50	0.077 / 0.077	**0.014 / 0.017**	0.050 / 0.058	0.274	0.522

Summarizing the data from all three results tables, *BayesSeg* gives the best performance whenever a well-defined list of stop-words is available. This is not surprising: the algorithm looks for the best segmentation among all possible segmentations. However, such an analysis, as well as Bayesian inference techniques, are computationally expensive. Fair and precise running time comparisons are not possible due to different programming languages. However, to give the reader an idea, the average running times of our algorithm (Python) and *C99* (Java) are about 0.5 seconds per document on the 30-50 dataset; *BayesSeg* (Java) requires on average 5.5 seconds per document. Furthermore, *BayesSeg* performs significantly worse when stop-words are not removed and this can be problematic since there is no canonical list. Therefore, adapting the algorithm to different domains (emails, tweets, IM, speech transcripts, etc.) or languages will most likely require manual adaptation. On the other hand, our algorithm proves extremely robust to stop-words and even performs best when stop-words are not removed. Furthermore, its relative performance compared to other algorithms improve as documents get longer. Those two facts can be explained by the statistical nature of the meaningfulness scores used for feature extraction. Indeed, these scores seem to get more reliable as more data becomes available.

5.4.2 Towards more realistic datasets

Limitations of synthetic datasets

Despite their wide-spread use for algorithm comparisons, synthetic datasets have several drawbacks.

First, the relative independence of consecutive segments in synthetic documents makes the segmentation task easy. Indeed, the segment transitions are quite abrupt because there is no natural flow of ideas between the different segments as should be expected from most textual documents, such as a book or a business report. Even more importantly, remembering that segmentation algorithms rely on lexical cohesion and word repetitions, detecting segment boundaries is artificially easy because:

- the different segments discuss independent topics and thus have different vocabulary,
- the different segments are written by different authors, who formulate ineluctably the same ideas using different words.

Second, Choi's synthetic dataset, which is in standard use for algorithm comparisons, contains very short segments ranging from 3 to 11 sentences. A similar dataset with segments up to 50 sentences long has been created but one must recognize that this number remains small and that it does not cover all potential uses of segmentation algorithms.

To overcome these issues, we will create our own datasets. Experiments in the remaining of this chapter are conducted with much larger documents, both in terms of longer average segment sizes and larger number of segments. To simulate more realistic datasets, genres will be varied and foreign languages will also be tested.

To avoid copyright restrictions and make the results reproducible, popular public domain books are used. All these works can be found on the Project Gutenberg's website [7] and on Wikisource [17]. The reference segmentations are taken as the division created by the author into chapters, books or volumes. The default implementation of each algorithm is used: *C99* and *BayesSeg* each use their own stop-words list while our algorithm does not filter stop-words.

Short stories dataset

Let us start our experiments with books containing short stories. Each segment being a different story, these give non-ambiguous reference segmentations. Styles and languages are varied and some books contain stories from a single author while some others contain stories from several authors. More precisely, the books used are:

- "Complete Original Short Stories" by Guy de Maupassant (English translation, Gutenberg book #3090).

- "Dubliners" by James Joyce (Gutenberg book #2814).
- "Fairy Tales" by Hans Christian Andersen (English translation, Gutenberg book #32571).
- "Famous Modern Ghost Stories" by several authors (Gutenberg book #15143).
- "Great English Short-Story Writers" by several authors (Gutenberg book #10135).

Some statistics regarding the different books are presented in Table 5.5. Numerical results are presented in Table 5.6.

Table 5.5: Short stories dataset statistics. Document and segment sizes are given in number of sentences.

	Document size	Number of segments	Average segment size	Segment size range
Maupassant	28476	180	158	29-700
Dubliners	4691	15	313	116-1005
Fairy Tales	2851	20	143	36-566
Ghost stories	5775	15	385	77-1023
Great English	3443	13	265	118-524

Table 5.6: WindowDiff scores (using Lamprier et al.'s correction) on the short stories dataset. Processing times are given between parentheses.

	C99	BayesSeg	Our algorithm	Uniform	Random
Maupassant	N/A[a]	N/A[a]	**0.236** (112s)	0.475	0.511
Dubliners	0.403 (8s)	0.333 (2430s)	**0.250** (25s)	0.485	0.556
Fairy Tales	0.331 (4s)	**0.133** (452s)	0.234 (8s)	0.500	0.508
Ghost stories	0.259 (15s)	0.242 (2603s)	**0.218** (47s)	0.497	0.557
Great English	0.247 (6s)	**0.001** (1508s)	0.150 (20s)	0.491	0.503

[a] Execution failure due to insufficient memory (tested with 12GB of RAM).

Analyzing the results from Table 5.6, the performance of all algorithms seems to be degraded as compared to results on the synthetic datasets but that is to be expected when testing on a more realistic and more difficult dataset. *BayesSeg* and our algorithm are the most accurate and stable as they consistently rank first or second on all documents, our algorithm being best on 3 out of 5 inputs. *C99* always comes third, sometimes well behind the first two algorithms.

Knowing that none of the algorithms were particularly optimized for this use case, two important details are also worth noting. Both *C99* and *BayesSeg* failed on one input document due to high space complexities which entail high RAM consumption when dealing with large inputs (tests were made with 12GB of RAM). Even on the largest documents, our algorithm never required more than 512MB of RAM to complete.

Furthermore, omitting the trivial baseline algorithms, *C99* is the fastest algorithm while our algorithm comes second, usually 2 to 3 times slower. Precise comparisons cannot be established since the former is implemented in Java while the latter is pure Python. However, the computational cost of *BayesSeg* explodes, making it too slow and hence impractical for large documents.

Novels dataset

Let us continue our experiments with novels. The reason for using novels is to work with consecutive segments truly forming a continuous logical flow. This is how most documents are likely to be in real use cases, and it is therefore important to check how the different algorithms behave. The books we are working with are:

- "Clarissa Harlowe; or the history of a young lady" by Samuel Richardson. This book is the concatenation of 9 volumes, that can be found as Gutenberg books #9296, #9798, #9881, #10462, #10799, #11364, #11889, #12180 and #12398.
- "Dracula" by Bram Stoker (Gutenberg book #345).
- "The Iliad" by Homer (English translation, Gutenberg book #2199).
- "The Metaphorphosis" by Franz Kafka (English translation, Gutenberg book #5200).
- "The Count of Monte Cristo" by Alexandre Dumas (English translation, Gutenberg book #1184).
- "War and Peace" by Leo Tolstoy (English translation, Gutenberg book #2600)

Statistics on the different books are presented in Table 5.7. The numerical results are shown in Table 5.8.

Table 5.7: Novels dataset statistics. Document and segment sizes are given in number of sentences.

	Document size	Segment types	Number of segments	Average segment size	Segment size range
Clarissa	38220	Volume	9	4247	3462-4869
Dracula	8398	Chapter	27	311	189-396
Iliad	6397	Book	24	267	164-368
Metamorphosis	797	Part	3	265	224-300
Monte Cristo	26263	Chapter	117	224	82-589
War and Peace	32014	Book	17	1883	562-3683

Table 5.8: WindowDiff scores (using Lamprier et al.'s correction) on the novels dataset. Processing times are given between parentheses.

	C99	BayesSeg	Our algorithm	Uniform	Random
Clarissa	N/A[a]	N/A[a]	0.294 (5322s)	**0.212**	0.531
Dracula	**0.394** (42s)	0.522 (6562s)	0.425 (70s)	0.566	0.582
Iliad	0.419 (33s)	0.513 (3673s)	**0.351** (36s)	0.396	0.570
Metamorphosis	0.427 (1s)	0.328 (30s)	0.127 (5s)	**0.088**	0.304
Monte Cristo	N/A[a]	N/A[a]	**0.344** (118s)	0.550	0.533
War and Peace	N/A[a]	N/A[a]	**0.340** (1451s)	0.402	0.421

[a] Execution failure due to insufficient memory (tested with 12GB of RAM).

As expected, the average WindowsDiff scores increase again (i.e., the algorithm accuracies decrease) with this dataset. Other than this, the experimental results validates most of the observations made on the short stories dataset.

Once again, our algorithm yields good results and seems pretty stable; only *C99* slightly outperforms it once (omitting the *Uniform* baseline for the moment). We do not see obvious reasons explaining why *BayesSeg* performs worse in the current setting.

C99 and *BayesSeg* fail to finish on 3 of the input documents which are notoriously long novels, making our algorithm the only one that finishes on these inputs. However, it should be pointed out that our algorithm becomes slow as the average segment size (i.e., the window size) increases. For example, the segmentation of "Clarissa" into volumes takes more than one hour whereas the segmentation of "The Count of Monte Cristo" into chapters is much faster, despite the lengths of the two books being of the same order of magnitude. The reason is simple: as the window size (in number of sentences) increases, the number of unique words appearing in each window increases so the computational time per window also increases. A number of modifications could make the algorithm run faster, for example:

- optimizing the current Python implementation.
- re-implementing the algorithm in a performance oriented language, like C++ or Java.
- parallelizing the computations.
- shifting the moving windows by k sentences or by a paragraph/section/etc. instead of moving it by one sentence at a time (see Algorithm 5.1, page 57). If the number of windows used decreases by a factor k, then the running time of the algorithm will also essentially be divided by k.
- filtering stop-words to reduce the input size as well as the vocabulary size.

It is also interesting to note that the *Uniform* baseline outperforms all "real" segmentation algorithms twice. This is simply due to the fact the corresponding documents have segments of relatively uniform sizes. It should be noted that our algorithm, due to the use of periodic boundary conditions, does not have any notion of start and end of a document, as opposed to the *Uniform* baseline. But more importantly, this emphasizes the importance of extrinsic evaluation for particular applications and in practical systems.

Evaluating an algorithm extrinsically means evaluating its performance on the system's end results. For example, if segmentation is used as a preprocessing step to improve the accuracy of a classifier (like in the case of classifying documents into sensitivity levels, e.g., top-secret), the best segmentation algorithm may not be the one that has the smallest WindowDiff score but instead the one that yields the highest classification accuracy. So, despite the *Uniform* baseline sometimes outperforming the segmentation algorithms in our intrinsic evaluation, real segmentation algorithms create segments that are ultimately much more topically consistent and thus that are likely to yield better results in practical systems.

Experimenting with common Western European languages

As a final experiment, let us investigate the behaviour of our algorithm on documents in foreign languages. To this end, we will rely on the book "Alice in Wonderland" written by Lewis Carroll. Given its wide popularity, public domain versions of this book are available in many languages. More precisely, we use the original version (Gutenberg book #11), a French translation (from Wikisource[3]), a German translation (Gutenberg book #19778), an Italian translation (Gutenberg book #28371) and a Spanish translation (from Wikisource[4]). Statistics on these different versions are available in Table 5.9 and the numerical results in Table 5.10.

Table 5.9: Alice in Wonderland dataset statistics. The reference segmentation is given by the 12 chapters. Document and segment sizes are given in number of sentences.

	Document size	Average segment size	Segment size range
English	1627	136	85-177
French	1720	143	104-191
German	1033	86	65-103
Italian	1588	132	89-173
Spanish	1881	157	99-218

Table 5.10: WindowDiff scores (using Lamprier et al.'s correction) on the Alice in Wonderland dataset. Processing times are given between parentheses.

	C99	BayesSeg	Our algorithm	Uniform	Random
English	0.470 (3s)	0.522 (88s)	**0.463** (3s)	0.515	0.549
French	N/A[a]	N/A[a]	**0.405** (4s)	0.435	0.475
German	N/A[a]	N/A[a]	0.428 (2s)	**0.214**	0.462
Italian	N/A[a]	N/A[a]	**0.306** (5s)	0.504	0.490
Spanish	N/A[a]	N/A[a]	**0.397** (4s)	0.473	0.525

[a] Implementation not adapted to languages other than English.

The only modification required by our algorithm to deal with the different languages is the stemming algorithm used internally, and stemmers for many languages are easily available thanks to the Snowball language [108]. The good results in Table 5.10 show that our algorithm can be easily and quickly adapted to a variety of languages as long as stemming algorithms as well as tokenization algorithms are available. Tokenization

[3]Downloaded from `https://fr.wikisource.org/wiki/Alice_au_pays_des_merveilles`

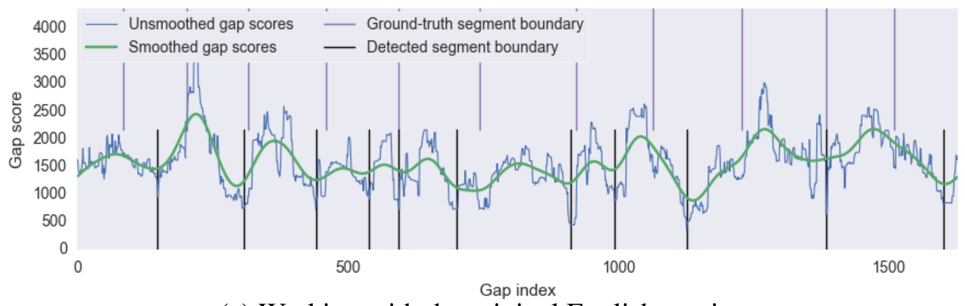
[4]Downloaded from `https://es.wikisource.org/wiki/Las_aventuras_de_Alicia_en_el_Pa%C3%ADs_de_las_Maravillas`

is easy for most European languages but can prove very difficult in some others. For example, Chinese is notoriously hard to tokenize since words are not white-space separated.

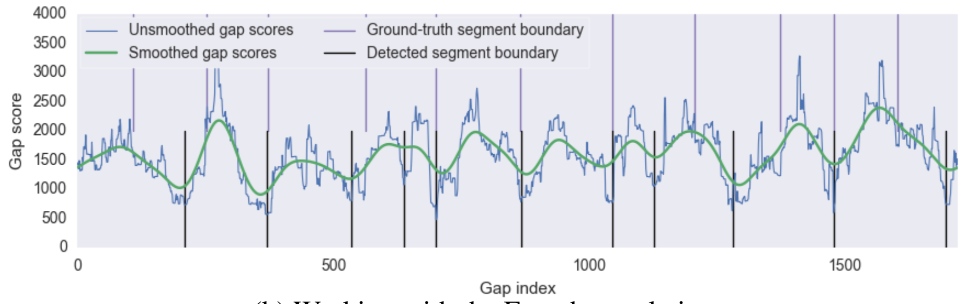
Figure 5.10 shows the gap scores obtained when segmenting "Alice in Wonderland" and its various translations, and it highlights two interesting facts.

First, one can see that both the gap scores and the output segmentations look very similar in the different languages available, despite some obvious translation differences (e.g., the number of sentences in the German translation is about two-thirds the number of sentences in the other versions of the book). This further strengthens the claim that our algorithm can easily be adapted to many languages, and this with an accuracy similar to the one obtained on English documents.

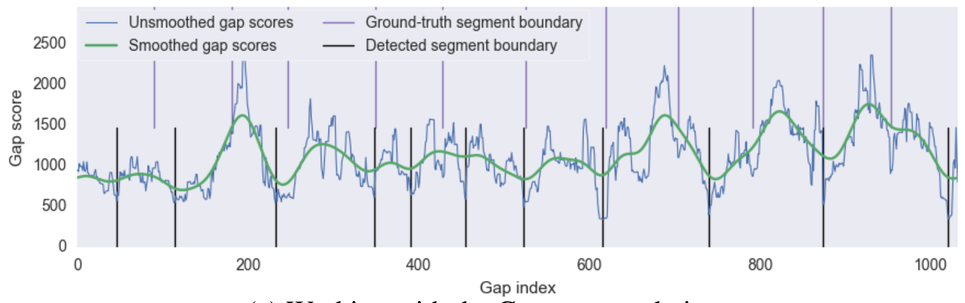
Second, the plots also explain why relatively poor results are obtained, that is, WindowDiff scores in the range $0.3 - 0.5$. Indeed, the end of the first chapter seems to be a region of high lexical cohesion and the segmentation algorithm consistently outputs a segment boundary in the middle of the second chapter... Why such a discrepancy? Reading the book provides an answer. The first chapter ends rather abruptly in the middle of Alice's time in the rabbit hole, without any big change in the storyline. However, our algorithm outputs a boundary later when Alice, still in the rabbit hole, starts crying, her tears forming a pool that allows her to enter the fantasy garden. And the location of the boundary seems quite sensible as the vocabulary suddenly changes at this time. The decision of an author to start and end chapters might take into account other aspects than just the story, which strengthens again the argument in favour of extrinsic evaluation whenever possible.



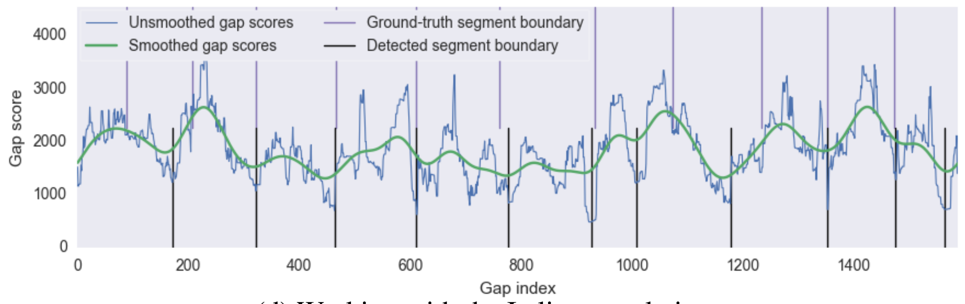
(a) Working with the original English version



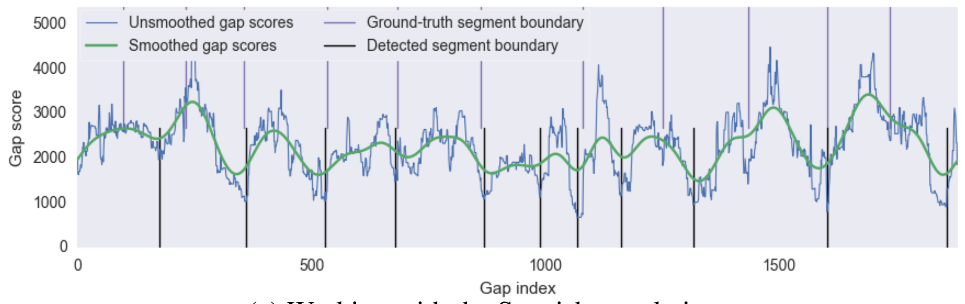
(b) Working with the French translation



(c) Working with the German translation



(d) Working with the Italian translation



(e) Working with the Spanish translation

Figure 5.10: Gap score plots of "Alice in Wonderland" and European translations, obtained with our algorithm.

5.5 Conclusion

Automatic text segmentation algorithms can be used to improve the performance of many text mining tasks, by splitting long documents into topically consistent segments and thus allowing subsequent mining algorithms to work at a finer and more sensible scale.

In this chapter, we introduced a novel text segmentation algorithm that is completely unsupervised. This algorithm takes a single input parameter, the desired number of segments, which allows to segment a document at different scales, from small paragraphs to large sections. All this is done by looking only at the document at hand without requiring any training or any external source of knowledge.

We performed an extensive evaluation and compared our algorithm with several state-of-the-art approaches. On small synthetic documents, our algorithm gives performances that are similar to the state-of-the-art. On longer and more realistic documents, our algorithm clearly outperforms the other algorithms on a wide range of inputs. Indeed, we showed that our algorithm has excellent accuracy and that it is both fast and extremely robust to stop-words, making it generic and able to deal with different genres, domains and jargons. Easy adaptation to several other Western European languages was also demonstrated.

Finally, our evaluation hinted at the limits of intrinsic evaluation; in any concrete system, it is always better to measure the performance of a segmentation algorithm as the impact on the system's end results rather than on a given standard metric. However, we believe that the algorithm we proposed is a strong candidate for a wide range of inputs and applications.

Chapter 6

Automatic Summarization

The goal of automatic summarization is to condense large documents into concise forms that retain the most important information. Summarization is a widespread need in text mining, so summarization algorithms found their way into many systems. The most common use of summaries is for rapid relevance assessment, as users often need to be able to quickly decide whether a document, or set of documents, is relevant to their need and worth reading. Summarization can go further and sometimes even seeks to replace the original documents, since with information overload it is not always possible to read large quantities of text. Examples include the creation of executive summaries and summaries for technology intelligence (that is, to keep track of what competitors do). But summarization can also be used as a pre-processing step, to improve the speed or performance of subsequent algorithms such as document clustering [55].

In this chapter, we extend the automatic summarization algorithm introduced by Balinsky et al. in [24, 25] and perform an extensive evaluation of the new algorithm. The chapter starts with a review of the state-of-the-art approaches to summarization in Section 6.1. The proposed summarization algorithm is then presented in Section 6.2. The evaluation methodology is introduced in Section 6.3 and Section 6.4 presents and discusses the empirical results. Section 6.5 concludes the chapter.

A summarization library has been developed as part of this work so that the proposed algorithm can be easily integrated in any text mining system. A Graphical User Interface (GUI) has also been designed to make experimentation easy. Both are described in Appendix A.3, page 113.

6.1 A brief literature review

Information overload is an ancient issue so very naturally, research on summarization has started early along with the development of computers. It has been a large research area ever since; let us briefly review it.

First and foremost, several intersecting categories of summaries exist depending on the systems and users requirements:

- A *generic* summary is aimed at a broad readership and does not focus on any particular aspect of the text, as opposed to a *query-focused* summary which takes into account specific information requirements to specialize the output summary. As an illustration, a query-focused summary of an encyclopedic article on William Shakespeare could concentrate on the critical reception of the author's work while a generic summary would incorporate all important aspects of his life and work.
- A summary can be created from one or several input documents. Those two tasks are respectively referred to as *single-document* and *multi-document* summarization. Combining several sources of information, such as a cluster of news articles about the same event, allows summaries to be more comprehensive. However, the main difficulty arising in this setting is the need to filter highly redundant information.
- *Extractive* summarization returns the concatenation of the most important parts from the input text as the resulting summary. These "parts" are most of the time sentences, but most algorithms can also work with clauses or paragraphs. *Abstractive* summarization is a more complex process that paraphrases the input to create summaries that are both more natural and more compact. However, due to the difficulties of understanding and generating natural language, abstractive systems are in practice usually limited to relatively shallow post-processing of extractive summaries, as in [53].

In this chapter, our attention will be focused on the simplest type, namely generic, single-document, extractive summarization. One of the earliest approaches to this problem was proposed by Luhn in 1958 [82]. It relies on the intuitive idea that frequent words in a document are important since they are descriptive of the document topics. The set of such "topic words" is determined by removing both rare and non-content words. To this end, two pre-determined thresholds can be used, for low and very high frequency words, as illustrated in Figure 6.1. Alternatively, a list of stop-words can be used in place of the high frequency threshold. Sentences that contain many topic words are then used to build the summary. This simple approach has inspired many recent algorithms. For example, rather than using raw frequencies, the SumBasic algorithm [100] estimates word probabilities from the input text. Each sentence is then associated

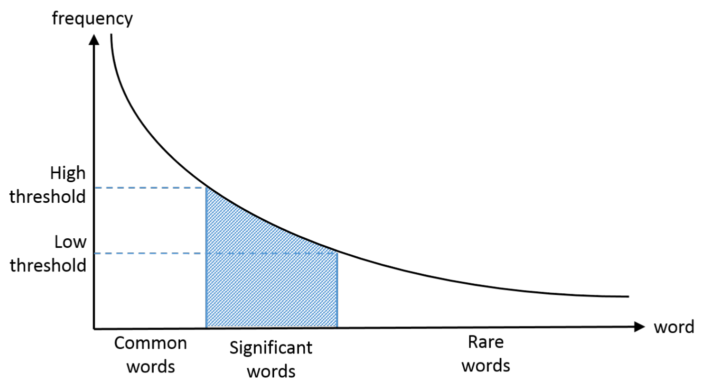


Figure 6.1: The use of frequency thresholds to determine topic words [82]. Such topic words are then used to identify important sentences. The x-axis represent words ranked in decreasing order of frequency.

a score equal to the average probability of the words it contains. When corpus statistics are available, the use of statistical hypothesis testing [76] or measures inspired from TF-IDF [54, 110] are also popular to determine important words and, in turn, important sentences.

Those frequency-based approaches perform a shallow analysis. Many algorithms doing a deeper and richer semantic analysis have been proposed. A relatively simple way to incorporate semantic knowledge is to group conceptually related words together. For example, the words "cell", "biology" and "organism" are very closely related but a simple frequency analysis will miss their semantic relation. Barzilay and Elhadad [28] proposed the use of lexical chains (i.e., sequences of related words). Semantic relations between words are determined using the WordNet thesaurus [94] and lexical chains are assigned a score based on their lengths and on the number of distinct words they contain. The summary is built by including one sentence from each highly scored chain, thus minimizing the redundancy of information. Gong and Liu [54] proposed the use of Latent Semantic Analysis (LSA) to determine semantic relations automatically. The algorithm constructs a matrix A, in which entries A_{ij} are TF-IDF weights for the word i in the sentence j (for computing IDF terms, each sentence is considered as a "document"). Using Singular Value Decomposition (SVD), each column of A is projected into the singular vector space, where each dimension is a linearly independent latent feature which captures relations between terms and, hopefully, a topic from the input document. The magnitude of the singular value associated with each dimension can be considered as a measure of its importance (i.e., the importance of the associated latent feature). Finally, the most representative sentence for each of the most important latent features is included in the summary.

A third important class of algorithms creates graph representations of the input documents. TextRank [92] and LexRank [49], which are relatively similar, are two popular examples. In both algorithms, the first step constructs a highly connected graph from the input document, where each sentence is a vertex in the graph and edges represent the similarity between pairs of sentences. The measure of similarity for TextRank is a normalized measure of content overlap:

$$Sim_{TextRank}(s_i, s_j) = \frac{|s_i \cap s_j|}{\log(|s_i|) + \log(|s_j|)} \tag{6.1}$$

where s_i and s_j are two sentences, defined in this context as sets of words. The preprocessing steps are not mentioned explicitly but they are likely to contain stop-words filtering and/or syntactic filtering to avoid the similarity scores from being dominated by non-important words. LexRank uses a more traditional cosine similarity with TF-IDF weights:

$$Sim_{LexRank}(s_i, s_j) = \frac{\sum\limits_{w \in s_i, s_j} \text{TF-IDF}(w, s_i) \times \text{TF-IDF}(w, s_j)}{\sqrt{\sum\limits_{w \in s_i} \text{TF-IDF}(w, s_i)^2} \sqrt{\sum\limits_{w \in s_j} \text{TF-IDF}(w, s_j)^2}} \tag{6.2}$$

where s_i and s_j are the bag-of-words for the two sentences. Then, a centrality measure is used to rank the nodes of the graphs (i.e., sentences) by order of importance and the most important sentences are used as summary. Both algorithms use an adaptation of the PageRank algorithm [32] invented by Page and Brin for determining the importance of pages on the web.

Finally, supervised learning algorithms can also be trained to compute sentence importance. Features such as position of the sentence in the original text, position of the sentence in the paragraph, sentence length and presence of predefined cue phrases are popular and often effective [71, 120, 135]. Measures of word importance are also usually incorporated in the models, using for example one of the methods discussed above. However, the need for gold-standard summaries [120, 35] makes supervised learning more difficult than traditional approaches, while not always leading to superior performance. More precisely, machine learning approaches have been shown to work best for domain or genre specific summarization [98].

6.2 Proposed algorithm

In this section, we extend the summarization algorithm introduced by Balinsky et al. in [24, 25]. It is a graph-based algorithm that follows the most common approach to summarization: after preprocessing, the first step consists in identifying important words in the document. These keywords are then used to build a graph, which is in turn used to identify the most important sentences. Let us start by briefly reviewing the keyword and sentence extraction procedures proposed in [24, 25].

6.2.1 Preprocessing

Input documents may be given as lists of sentences, especially for evaluation purposes, in which case sentence tokenization is not necessary. When it is not the case, a simple regex-based tokenizer can be used, considering the characters ".", "!" and "?" as sentence boundaries. Then, each sentence is split into a list of words. A regex-based tokenizer can also be used, taking all non-alphabetical characters as word delimiters. Then all words are downcased and the Porter stemming algorithm is applied. In [24, 25], Balinsky et al. verified that the Helmholtz principle approach automatically filters most stop-words on long documents, or said differently, that stop-words filtering was not necessary. As this may not hold on shorter documents, the effect of stop-words on shorter inputs will be carefully studied in this chapter.

6.2.2 Keyword extraction

Once the input document is preprocessed, the most important words are then identified. Keywords extraction is performed in an unsupervised way using only the document to summarize. Let $\mathcal{P}$ denote a family of parts of a document D. Elements of $\mathcal{P}$ can be paragraphs or sections of D (possibly computed with the segmentation algorithm described in Chapter 5) if such logical units are available or, more generally, several consecutive sentences. In the latter case, these elements can be overlapping. The meaningfulness score $meaning(w, P, D)$ (see Equation (3.11), page 22), derived from the Helmholtz principle, is used as a measure of *local* word importance; here w denotes a word and P an element of $\mathcal{P}$.

But more than these local keywords, a set of keywords for the entire document D is required. To define these *global* keywords, let us start by defining ϵ-meaningfulness:

$$w \text{ is } \epsilon\text{-meaningful inside } P \Leftrightarrow meaning(w,P,D) \geq \epsilon \tag{6.3}$$

Now, a word w is considered a global keyword for the document if this word is ϵ-meaningful in at least one part $P \in \mathcal{P}$. This set of global keywords is denoted by $MeaningfulSet(D,\epsilon)$ and it is thus defined as:

$$MeaningfulSet(D,\epsilon) = \{w : \max_{P\in\mathcal{P}} meaning(w,P,D) > \epsilon\} \tag{6.4}$$

ϵ is a parameter used to vary the size of this global keywords set, typically chosen strictly positive as we are only interested in meaningful words. Note that it is possible to rank the words from $MeaningfulSet(D,\epsilon)$ by order of importance, according to their $\max_{P\in\mathcal{P}} meaning(w,P,D)$ value, as illustrated in Table 6.1.

Table 6.1: Ranking words of a fictitious document D according to their $\max_{P\in\mathcal{P}} meaning(w,P,D)$ value. The set $MeaningfulSet(D,\epsilon=0)$ corresponds the top part of the table, up to the stem 'find'**.**

Porter stem w	$\max_{P\in\mathcal{P}} meaning(w,P,D)$	Importance	Original words
art	4.591	Most	art, arts
claim	4.275		claims, claimed
...	...	↕	...
find	0.021		finding, findings
data	0.000		data
...	...		...
year	-8.375	Least	years

6.2.3 Sentence extraction

The next step in the document modelling from [24, 25] is the construction of a small-world graph. The term *small-world* refers to the qualities observed on social networks (in particular the small average path length) and was popularized by Milgram's experiment in 1967 [93]. More precisely, a network is said to be a small-world if it has the following three properties (see Chapter 20 in [44] for more details):

- *a small number of edges:* small-world graphs are sparse. Intuitively, individuals in a social network typically know a few hundreds of people, which is a very small number in comparison to the size of the network, typically in millions of individuals.

- *a high level of transitivity:* transitivity is best translated in words by the phrase "the friend of my friend is likely to also be my friend". A high transitivity expresses the tendency to observe strong local communities and relatively few long distance connections.

- *a short average distance:* despite being sparse and essentially made of local communities, small-world graphs have the surprising property that randomly selected pairs of nodes are, on average, at a small distance from each other. This property was first shown by Milgram [93] and gave rise to the famous phrase "the six degrees of separation". This property has been empirically verified on a number of more recent social graphs, such as Facebook [129].

Small-world networks have recently received a lot of attention from researchers as a large number of natural, technological and social networks were found to exhibit such properties [133, 132].

In our case, such a small-world graph can be built based on the extracted keywords as follows. Let us denote by $s_1, s_2, \ldots, s_n$ the sequence of consecutive sentences in the document D. The graph $Gr(D, \epsilon)$ has the sentences $\{s_i : i \in [1..n]\}$ as its vertex set. Sentences usually create a flow of information in a text, so an edge is added between every pair of consecutive sentences (s_i, s_{i+1}); this ensures connectivity of the graph, as shown in Figure 6.2a. An edge is also added between every pair of sentences sharing a least one common keyword, that is to say a word from $MeaningfulSet(D, \epsilon)$. This has the effect of creating many short-distance and a few long-distance references. Hence ϵ also controls the number of edges in the graph: when ϵ is too big, the set of keywords is empty and the graph degenerates to a path graph; when it is too small, the graph becomes a large random graph. In between these two extreme cases and for some carefully chosen set of keywords, the graph exhibits a small-world topology. Balinsky et al. empirically determined that the transition into a small world occurs, for large documents, when the number of edges is between $3n$ and $4n$ [23]. We consider the *target number of edges* (say, $3n$ or $4n$) as a parameter of the algorithm. To build such a small-world graph with a predefined target number of edges, the set of all meaningful words $MeaningfulSet(D, \epsilon = 0)$ is first computed and sorted by decreasing level of meaningfulness. Then, each keyword is considered in turn, starting with the most meaningful one, and an edge is added between every pair of sentences containing that

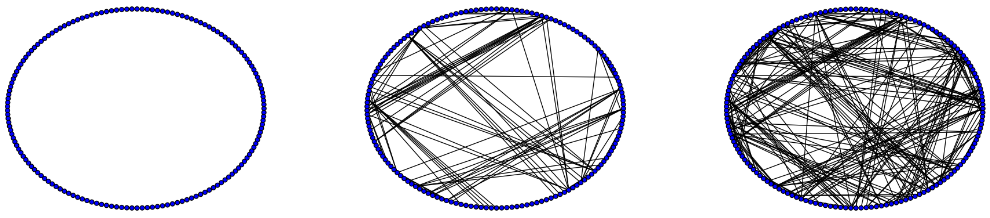

(a) Initial path graph (b) After adding a few edges (c) Final small-world graph

Figure 6.2: Small-world building procedure.

keyword. The procedure stops when the target number of edges in the graph is reached (see Figure 6.2c).

Finally, the most important sentences (i.e., the most central vertices) are extracted from the graph. Graph and social network theories provide many centrality measures in order to identify the most important, or central, nodes in a graph [101]. Common examples include the degree, betweenness and eigenvector centrality measures; these and several more measures will be described in more details in Section 6.2.6.

6.2.4 Complete description of the original algorithm

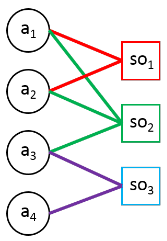
Algorithm 6.1 gives the pseudo-code for the complete algorithm proposed by Balinsky et al [24, 25].

Algorithm 6.1: Original summarization algorithm [24, 25].

```
Preprocess the document $D$.
Compute $MeaningfulSet(D, \epsilon = 0)$ [see Equation (6.4) page 81] and
sort it by decreasing level of meaningfulness.
Create a graph $G$ with $n$ vertices.
While the target number of edges in $G$ is not reached and not
all keywords have been used:
  Pick the next keyword in the sorted $MeaningfulSet(D, 0)$.
  For each pair of sentences that both contain this keyword:
    Add an edge between the two corresponding vertices.
Rank the sentences by decreasing importance using a centrality
measure.
Return the most important sentences as summary.
```

6.2.5 Extending the original algorithm using affiliation networks

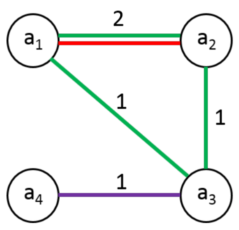
Balinsky et al. [24, 25] justified their graph construction procedure using *affiliation networks*, which were introduced by Lattanzi and Sivakumar [73]. Affiliation networks are bipartite graphs with two types of nodes: *actors* and *societies*. Each actor can be affiliated to any number of societies and edges of the graph denote the membership of an actor to a society, as illustrated in Figure 6.3a. One can then create a social graph based on society co-membership as follows. Let us start by defining the *affiliation matrix* A as the binary matrix where the coefficients A_{ij} denote the membership of the i-th actor to the j-th society. An example affiliation matrix is shown in Figure 6.3b. The adjacency matrix of the corresponding co-membership social graph is simply computed as $M = A \times A^T$ as illustrated in Figure 6.3d. Thus the coefficient M_{ij} is the result of the dot product between the i-th and j-th rows of A, which respectively contain the membership information of the i-th and j-th actors. As a result, the edge weights from the co-membership graph correspond to the number of societies that actors have in common. An example social graph is shown in Figure 6.3c. We will keep denoting by A and M, respectively, the affiliation and co-membership adjacency matrices in the remainder of the chapter.



(a) Affiliation network

$$A = \begin{array}{c|ccc|} & so_1 & so_2 & so_3 \\ \hline a_1 & 1 & 1 & 0 \\ a_2 & 1 & 1 & 0 \\ a_3 & 0 & 1 & 1 \\ a_4 & 0 & 0 & 1 \end{array}$$

(b) Affiliation matrix



(c) Co-membership graph

$$M = A \times A^T = \begin{array}{c|cccc|} & a_1 & a_2 & a_3 & a_4 \\ \hline a_1 & 2 & 2 & 1 & 0 \\ a_2 & 2 & 2 & 1 & 0 \\ a_3 & 1 & 1 & 2 & 1 \\ a_4 & 0 & 0 & 1 & 1 \end{array}$$

(d) Co-membership adjacency matrix

Figure 6.3: An example of affiliation network and the corresponding social graph. a_is represent actors and so_js represent societies.

Coming back to summarization, sentences play the role of actors and the keywords identified by $MeaningfulSet(D, \epsilon)$ represent the societies. All sentences that share a keyword belong to the same society and thus are linked by edges (in other words, they form a clique). The construction of the initial path graph is not directly explained by affiliation networks, though it can be seen as adding $n-1$ societies, one for each pair of consecutive sentences (n denoting the number of sentences in the input document). For simplicity, the adjacency matrix M is computed in practice from the affiliation matrix A without these additional societies. Then, all off-diagonal coefficients (i.e., of the form $M_{i,i-1}$ and $M_{i,i+1}$) that are 0 are simply set to 1.

In the original summarization algorithm proposed by Balinsky et al. [24, 25], the graph is unweighted and undirected so its adjacency matrix M is a symmetric binary matrix. In the framework of affiliation networks, this corresponds to using a binary affiliation matrix A, where A_{ij} coefficients denote whether the keyword w_j appears in sentence s_i, and then binarizing the adjacency matrix computed as $M = A \times A^T$. However, the affiliation and adjacency matrices do not need to be binary. A number of alternatives can potentially be used and we propose to extend Balinsky et al.'s model by defining new affiliation matrices:

- The binary matrix A^{bin}, where A^{bin}_{ij} denotes whether the keyword w_j appears in sentence i.
- The local score matrix A^{local} where A^{local}_{ij} is the local meaningfulness score $meaning(w_j, P, D)$ of the keyword w_j inside P, where P is the part encompassing the sentence s_i. If meaningfulness scores are computed using sliding windows, the local score is the maximum score got over all windows (i.e., parts P) encompassing s_i. Also, if the meaningfulness score is negative or if the keyword w_j does not appear in the sentence s_i, A^{local}_{ij} is set to 0.
- The global score matrix A^{global} where A^{global}_{ij} is the "global" meaningfulness score of the keyword w_j inside the sentence s_i, defined as:

$$A^{global}_{ij} = \begin{cases} \max_{P \in \mathcal{P}} meaning(w, P, D) & \text{if } w_j \text{ appears in } s_i \\ 0 & \text{otherwise.} \end{cases} \tag{6.5}$$

Based on the affiliation matrices A^{bin}, A^{local} and A^{global}, we can create several new models for building small-world graphs. Multiplying any of the proposed affiliation matrix by its transpose yields a symmetric matrix because $(A \times A^T)^T = (A^T)^T \times A^T = A \times A^T$. Since by convention the (i, j)-th coefficient of any adjacency matrix is the weight of the edge going from the i-th vertex to the j-vertex, a symmetric adjacency matrix corresponds to an *undirected* graph because edges weights between any pair of vertices are the same in both directions. But we can also multiply two different affiliation matrices together, which in most cases results in a non-symmetric adjacency matrix and thus a *directed* graph.

We propose the five small-world models described below. For interpreting these, it helps to keep in mind that if $M = A_1 \times {A_2}^T$, then the (i, j)-th coefficient of M is the dot product of the i-th row of A_1 (describing the keywords appearing in the sentence s_i) and the j-th row of A_2 (describing the keywords appearing in s_j).

1. **Undirected Counts (UC):** this model is computed as $M^{UC} = A^{bin} \times A^{bin^T}$. Coefficients of M^{UC} denote the number of (unique) keywords in common between pairs of sentences. The more keywords two sentences share, the higher the associated edge weight is. To ensure the graph is connected, off-diagonal coefficients that are 0 are set to 1.

2. **Undirected Binary (UB):** this is the original model with binary weights [24, 25], the adjacency matrix M^{UB} being obtained by binarizing M^{UC}.

3. **Undirected Local Meaning (ULM):** this undirected model is computed as $M^{ULM} = A^{local} \times A^{local^T}$. Thus the (i, j)-th coefficient of M^{ULM} is equal to:

$$M_{ij}^{ULM} = \sum_{k=1}^{|MeaningfulSet(D,\epsilon)|} A_{ki}^{local} \times A_{kj}^{local}. \tag{6.6}$$

   In words, each keyword shared between two sentences contributes to the weight of the corresponding edge by adding the product of their local meaningfulness scores. The rationale behind this model is that the more *locally* meaningful a keyword is, the more it should contribute to the edge weights of the associated neighbourhood of the graph. Note that keywords must be locally meaningful in both sentences in order to contribute to the corresponding edge weight.

   To ensure the graph is connected, consecutive sentences that are not connected are given an edge weight set to the lowest admissible value: the square of the smallest strictly positive entry of A^{local}.

4. **Undirected Global Meaning (UGM):** this undirected model is computed as

$M^{UGM} = A^{bin} \times A^{global^T}$. Thus the (i,j)-th coefficient of M^{UGM} is equal to:

$$\begin{aligned} M^{UGM}_{ij} &= \sum_{k=1}^{|MeaningfulSet(D,\epsilon)|} \mathbb{1}\left[w_k \in s_i\right] A^{global}_{kj} \\ &= \sum_{k=1}^{|MeaningfulSet(D,\epsilon)|} \mathbb{1}\left[w_k \in s_i \cap s_j\right] \max_{P \in \mathcal{P}} meaning(w,P,D), \end{aligned} \quad (6.7)$$

where $\mathbb{1}\left[p\right]$ is an indicator function that is equal to 1 if the predicate p is true and 0 otherwise. Despite being the result of the product of two different matrices, M^{UGM} is symmetric as clearly visible in Equation (6.7) and therefore it represents an undirected graph. The rationale behind this model is that the more a keyword is *globally* meaningful, the more it should contribute to the edge weights of the graph.

To ensure the graph is connected, off-diagonal coefficients that are 0 are set to the lowest admissible value, which is the smallest strictly positive entry of A^{global} or the smallest global meaningfulness value from any keyword in $MeaningfulSet(D,\epsilon)$.

5. **Directed Local Meaning (DLM):** the last model builds a directed graph, which is computed as $M^{DLM} = A^{bin} \times A^{local^T}$. As such, the (i,j)-th coefficient of M^{DLM}, indicating the weight of the edge from s_i to s_j, is equal to:

$$M^{DLM}_{ij} = \sum_{k=1}^{|MeaningfulSet(D,\epsilon)|} \mathbb{1}\left[w_k \in s_i\right] A^{local}_{kj}. \quad (6.8)$$

In words, this is the sum of the local scores in the sentence s_j, over keywords that appear in both s_i and s_j. The intuition behind this model is that shared keywords between a pair of sentences create two references, one in each direction, which strengths depend on local meaningfulness scores. A sentence that has high local meaningfulness values will have strong incoming weights.

To ensure the graph is connected, off-diagonal coefficients that are 0 are set to the lowest admissible value, which is the smallest strictly positive entry of A^{local}.

Now that we have proposed several ways to build small-world graphs from a document, we finally turn our attention to centrality measures, which we use in order to identify the most central (i.e., important) sentences to include in the summary.

6.2.6 Centrality measures

There are many possible ways to define node importance in a graph and very naturally, many different centrality measures exist (see Chapter 10 in [59]). However, most centrality measures are defined for unweighted and undirected graphs as these are the simplest and most prevalent type of graphs. So let us start by presenting these.

Centrality measures for unweighted and undirected graphs

The centrality measures for unweighted and undirected graphs that we use in this chapter are [59]:

- *Number of keywords:* this is not a centrality measure per se, but the simplest possible way to rank sentences is by counting the number of keywords each sentence contains. This is a rather intuitive idea that is commonly found in a way or another in the summarization literature [82, 100].

- *Degree centrality:* the degree centrality is simplest centrality measure where the importance of a vertex v is simply given by its degree:

  $$C_D(v) = degree(v). \tag{6.9}$$

  This centrality formalizes the idea that vertices with a large number of neighbours are the most important. Note that given our graph construction procedure, this is not the same as counting the number of keywords in each sentence. Indeed, a keyword appearing in n_s sentences adds $n_s - 1$ to the degree of each of these sentences (i.e., nodes).

  Though intuitive, the degree centrality is a purely local measure that completely ignores the wider topology of the network; the following other centrality measures build on more global notions of importance.

- *Closeness centrality:* the closeness centrality is the inverse of the average distance of a node to all other nodes in the network. With this definition of centrality, the most important nodes are the ones that can broadcast information to the entire network in a fast way.

  $$C_C(v) = \frac{|V| - 1}{\sum_{s \neq v \in V} distance(s, v)} \tag{6.10}$$

where $distance(s, v)$ denotes the distance (given by the shortest path) between s and v. Closeness is sometimes understood as independence [50], as the nodes closest to all other nodes are less dependent to intermediaries to get access to the information flow.

- *Betweenness centrality:* the betweenness centrality of a vertex v quantifies the control that a node has to manipulate information in a network:

$$C_B(v) = \sum_{s \neq t \neq v \in V} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{6.11}$$

where V denotes the vertex set, σ_{st} the number of shortest paths between s and t, and $\sigma_{st}(v)$ the number of shortest paths between s and t going through v. Thus, degrees with high betweenness centrality are on a large number of shortest paths and can easily alter information circulating in the network.

- *Eigenvector centrality:* the eigenvector centrality value of a node is proportional to the weighted average of the centrality values of its neighbours:

$$C_E(v_i) = \frac{1}{\lambda} \sum_{j \in \{1,n\}} M_{ij} C_E(v_j) \tag{6.12}$$

where λ is a constant [101]. In matrix form, the problem can be rewritten as $Mx = \lambda x$, where x is the centralities vector. It is easy to see that x is an eigenvector of the adjacency matrix M, and from the Perron-Frobenius theorem, it must be the eigenvector associated to the largest eigenvalue [101]. Intuitively, the eigenvector centrality takes into account the number of neighbours of a node but also the importance of those neighbours in the entire graph. Iterative methods are usually used for computations.

Adapting the centrality measures for weighted and/or directed graphs

Since some of the small-world models we proposed are weighted and/or directed small-world graphs, the centrality measures we just introduced need to be adapted to these types of graphs.

Before attempting to do so, it is important to understand the meaning given to edge weights. In all our models, large weights represent either high meaningfulness values or large numbers of shared keywords, and thus represent high similarities between sentences. Thus, the degree centrality can be naturally extended to weighted graphs, the weighted degree of a vertex being simply the sum of weights of all its adjacent edges. The most important sentences are therefore the ones with the highest degrees. Similarly, eigenvector centrality considers weights as a measure of importance and extends naturally to work with a weighted adjacency matrix. However, this is not the case for betweenness and closeness centralities which require some adaptation. Indeed, both rely on distances and shortest paths between vertices. Intuitively, our weights are the opposite of distances: high weights, or high similarities between vertices, should represent small distances. This is why edge weights need to be adapted for these two centralities. A number of monotonically decreasing transformations can be used. We choose the inverse function:

$$distance(w) = 1/w \tag{6.13}$$

where w denotes an edge weight. Thus high weights are transformed into small distances and vice versa. This function also transforms 0 weights (i.e., missing edges) into infinite distances, which is the expected behaviour.

Regarding directed graphs, the degree centrality can be computed as the sum of in-degrees (either weighted or unweighted). Betweenness and closeness centralities do not require adaptation other than taking into account edge directions while computing shortest paths. Finally, as important nodes have strong incoming links, the left eigenvector centrality must be used; that is, x must be a solution of the system $xM = \lambda x$ (see Chapter 21 in [86] for details).

6.2.7 Complete algorithm description

Let us conclude the section by presenting the high-level pseudocode for our complete algorithm.

Algorithm 6.2: Complete summarization algorithm.

```
Preprocess the document $D$.
Compute $MeaningfulSet(D, \epsilon = 0)$ [see Equation (6.4) page 81] and
sort it by decreasing level of meaningfulness.
Create a graph $G$ with $n$ vertices.
Create two empty affiliation matrices $A_1$ and $A_2$.
While the target number of edges in $G$ is not reached and not
all keywords have been used:
  Pick the next keyword in the sorted $MeaningfulSet(D, 0)$.
  Add one row to each affiliation matrix, with weights set
  according to the chosen model.
  Add the newly created edges to $G$.
Compute the adjacency matrix $M = A_1 \times A_2$.
Set the edge weights in $G$ according to the coefficients of M.
Add edges between consecutive sentences if necessary to ensure
the graph is connected.
Rank the sentences by decreasing importance using a centrality
measure.
Return the most important sentences as summary.
```

6.3 Evaluation methodology

6.3.1 Evaluation procedure

Summaries of different lengths can be created and as such, the desired length need to be an input to the summarization algorithm. Lengths can be a fixed number of sentences (say 5 or 10 sentences), or for comparison purposes, an approximate number of words since sentence lengths can range from a single word to several dozens. In the latter case, 100 and 200 words targets are common [75]. Some tolerance around that target number is obviously necessary to include only complete sentences but most evaluation metrics penalize summaries that deviate too much from the target number of words.

We will evaluate our algorithm on a variety of documents, both in genre and length. Balinsky et al. have tested their model on large documents [24, 25], made of several hundreds of sentences. However, short documents are extremely common (emails, web pages, etc.) and since small-world graphs are typically large, the concept of small-world becomes ill-defined in this setting. Therefore, one should check whether the algorithm still works on short documents.

Also, the algorithm will be compared with two state-of-the-art summarization algorithms:

- TextRank [92], an unsupervised approach fully described in Section 6.1, and
- CCS-NSA-v2 [120], a supervised learning algorithm based on Hidden Markov Models.

A popular baseline that simply outputs the first few sentences at the beginning of the document [2] will also be used. We denote it *First*. The reason for using this baseline is simple: it is often common to discuss the main points of a document in the introduction. For example, in the case of research publications, abstracts are actually short summaries created by the authors. Unsurprisingly, abstracts are good baseline summaries. Similarly, this baseline was shown to be very effective for news stories [2], as the main ideas are often briefly mentioned at the very beginning of such documents before being developed in the body of the story.

6.3.2 Evaluation metrics

Evaluating the quality of a summary is a complex task [85]. Most methods focus on content selection and work by comparing a candidate (automatic) summary to human summaries serving as references. However, creating such reference summaries is both expensive and time-consuming. Furthermore, several references are required for unbiased algorithm comparisons as there is no "best" summary; different human experts will create different, but equally good, summaries. The need for reference summaries also means that evaluating summaries at different lengths (say 50 words and 200 words) requires a set of references for each length, yet adding to the difficulty of creating a good evaluation corpus.

Historically, sentence recall, precision and F-measure have played an important role in summarization evaluation [28, 76]. However, the sentence granularity captures very little of the actual summary informativeness. In an attempt to solve this issue, Nenkova and Passonneau proposed an appealing alternative called the Pyramid method [99]. However this method requires a lot of manual work to identify the so-called "Summary Content Units", which are somewhat fuzzily defined as the important pieces of information that appear in the reference summaries. Fully automatic methods are therefore usually preferred.

Currently, the standard set of evaluation measures is called ROUGE and was developed by Lin [75]. ROUGE includes several measures based on overlapping units between a candidate summary and a set of gold-standards. ROUGE measures are fully automatic, work at the sub-sentential level and were shown to give excellent correlations with human judgments (Pearson's correlations up to 0.99) [75] on the Document Understanding Conference 2002 (DUC 2002) data [2]. Note that by working at the sub-sentential level (e.g., with word n-grams), ROUGE measures are suitable for evaluating both extractive and abstractive summaries.

We use the most common measure, ROUGE-N, that relies on word N-grams (i.e., sequences of N contiguous words) to compare a candidate summary c and a set of gold-standard summaries $R = \{r_1, \dots\}$. The formulae for the recall and precision versions of ROUGE-N are respectively presented on Equations (6.14) and (6.15).

$$\text{ROUGE-N}_{recall}(c, R) = \frac{\sum\limits_{r \in R} \sum\limits_{ngram \in r} \min\Big(f(ngram, r), f(ngram, c)\Big)}{\sum\limits_{r \in R} \sum\limits_{ngram \in r} f(ngram, r)} \tag{6.14}$$

$$\text{ROUGE-N}_{precision}(c, R) = \frac{\sum\limits_{r \in R} \sum\limits_{ngram \in r} \min\Big(f(ngram, r), f(ngram, c)\Big)}{|R| \times \sum\limits_{ngram \in c} f(ngram, c)} \tag{6.15}$$

where $f(ngram, s)$ is the frequency of the N-gram $ngram$ in the summary s. In plain words, the numerator of both formulae computes the number of overlapping N-grams between the candidate summary and the references. The denominators are simply normalization terms, included to scale values between 0 and 1. Note that it is easy to get higher recall and precision scores by using respectively longer and shorter candidate summaries. The F-measure version of ROUGE-N also exists and is the harmonic mean between recall and precision:

$$\text{ROUGE-N}_{fscore}(c, R) = 2\frac{\text{ROUGE-N}_{precision}(c, R) \times \text{ROUGE-N}_{recall}(c, R)}{\text{ROUGE-N}_{precision}(c, R) + \text{ROUGE-N}_{recall}(c, R)} \tag{6.16}$$

This is often the preferred version in order to penalize candidate summaries that are either too long or too short.

In the following, we use the F-score version of the ROUGE-1 metric, that is to say ROUGE-N_{fscore} working with unigrams ($N = 1$), as it was found to be robust on a variety of datasets [75] and it is the most commonly used metric in the literature. Note that ROUGE scores can be computed after stemming and/or stop words removal but it has been have shown to have minor impact on correlations with human judgements [75].

Therefore, all scores presented in this chapter are computed without any stemming or removal of stop words.

6.4 Experimental results

6.4.1 Comparative evaluation on the DUC 2002 dataset

In this section, we evaluate our algorithm on the standard evaluation dataset for summarization: DUC 2002 [2]. This dataset is made of 567 short news articles and we focus on the single-document generic summarization task, consisting in the creation of a 100-word summary for each news article.

In the original DUC competition, the summaries created by all competing systems were manually evaluated by comparison with two abstractive human-written summaries considered as gold-standards. However, no system was able to beat the baseline made of the first sentences of each article, so the single-document generic summarization task has been discontinued after 2002 [97].

More recently, research on evaluation has had significant progress and the evaluation framework ROUGE [75] introduced in 2004 helped to better compare systems performance so it has become the preferred evaluation tool. Therefore, the DUC 2002 dataset is still the standard dataset for such evaluations, despite its shortcomings. We use this dataset as a way to:

- perform a comparative evaluation with state-of-the-art algorithms.
- study the effect of the different parameters on the summarization results.
- study the behaviour of our algorithm with small documents. Indeed, small world networks are typically large graphs but the evaluation documents are a few tens of sentences long (only 3 documents, out of 567, that are longer than 100 sentences).

The numerical results are summarized in Table 6.2.

The best system according to the ROUGE-1 metric is CCS-NSA-v2 [120] (referred as System 28 in the original DUC 2002 competition) trained on a corpus of 119 documents. The *First* baseline remains a very good performer as only 3 systems from the original competition, out of 13, scored better. Precise numerical comparison with TextRank [92] is not possible as only the recall version of ROUGE-1 is reported in the

Table 6.2: Results on the DUC 2002 dataset, using ROUGE-1 F-score.

	F-Score
Automatic summarization systems:	
CCS-NSA-v2 (DUC 2002 winner)	0.4652
Our algorithm	0.4254
Baseline:	
First (sentences at beginning of document)	0.4577
Human experts:	
Best (expert G)	0.5379[a]
Worst (expert C)	0.4580[a]

[a] Expert summaries are evaluated against the other expert summary for the same document. Therefore, only one gold-standard is available when evaluating experts.

original article. The problem with recall is that longer summaries tend to get higher scores.

The effects of the parameters on our algorithm are presented in Table 6.3. In the remainder of this section, we present and analyze these experimental results.

On stop-words filtering

One of the main strengths of the Helmholtz approach to keywords extraction is the simplicity and speed of preprocessing, allowing fast treatment of large amounts of data. On large documents, the only preprocessing step performed is stemming, since stop-words are automatically excluded from the set of keywords on large documents, unless they are actually meaningful [22]. For example, in a chapter on determiners of an English grammar book, words like "the" and "a" are important and will be considered as such by our algorithm. This may not be true for short documents. Indeed, if a stop-word occurs frequently, but only locally as it may be common in short documents, our algorithm considers this as unusual behaviour; thus this word becomes a keyword. We can hypothesize that considering such non important words as keywords may have adverse effects on the results.

To check whether stop-words are problematic with short documents, we compare the results with and without stop-words filtering. In the first case, we do not remove stop-words at preprocessing time so that levels of meaningfulness are computed with

Table 6.3: Effect of our algorithm's parameters on the results on the DUC 2002 dataset, using ROUGE-1 F-score. The first row indicates the best set of parameters; in all other rows, the value of the parameter under study appears in blue.

Stop-words	Window size	Nb. of edges	Centrality	Model[a]	F-Score
127	4	∞	Degree	UB	**0.4254**
None	4	∞	Degree	UB	0.4172
3479					0.4206
127	2	∞	Degree	UB	0.4219
	3				0.4244
	5				0.4224
127	4	$2n$	Degree	UB	0.4138
		$3n$			0.4227
		$4n$			0.4245
127	4	∞	Nb. keywords	UB	0.4199
			Betweenness		0.4107
			Closeness		0.4218
			Eigenvector		0.4186
127	4	∞	Degree	UC	0.4218
				ULM	0.3842
				UGM	0.4000
				DLM	0.4047

[a] UB: Undirected Binary
UC: Undirected Counts
ULM: Undirected Meaning
UGM: Undirected Meaning
DLM: Directed Meaning.

all words; otherwise stop-words filtering would affect the size of the parts $P \in \mathcal{P}$ and thus the parameter N_{adapt} of the meaningfulness scores (see Equation (3.8) on pages 20 for the definition). Instead, stop-words are filtered only when building the graph, discarding all keywords belonging to a predefined list. Delaying stop-words filtering as such also prevents certain sentences from having all their constituent words filtered (e.g., "To be or not to be."). The stop-words list we use is minimized, with 127 words, mainly determiners and pronouns. It results that stop-words filtering has a big impact on the results: it improves the final F-score by almost 1%, from 0.4172 to 0.4254 (see Table 6.3).

Since stop-words removal is important and given that there is no universal list of stop-words, we also investigate whether we can use levels of meaningfulness to automatically create our own a list of stop-words. We proceed by concatenating all 567 documents of the DUC 2002 dataset, considering each news article as a container and running a modified keywords extraction procedure. We simply compute the set of *non-meaningful words*:

$$\{w : \max_{P \in \mathcal{P}} Meaning(w, P) < 0\} \tag{6.17}$$

We extract a list of 3479 words that are not meaningful in any document and consider them as stop-words; "the", "of", "a", "drive" and "practic" are a few examples of extracted words. Using these words as stop-words slightly improves the F-score to 0.4206, compared to a F-score of 0.4172 without stop-words filtering, but the results are still inferior to the ones when we use the predefined list of 127 words (F-score of 0.4254). However, we can expect much better improvements when dealing with non natural language processing applications, where non-content words are easier for an algorithm to identify (e.g., instructions like "*SELECT*" or "*INSERT*" for log files of SQL queries).

Parameter selection

We have two important parameters to set, namely the *window size* (that is to say the number of consecutive sentences used to define the family of parts $\mathcal{P}$ for each news article) and the *target number of edges* (i.e., the number of edges to insert in the graph). Additionally, we study the effect of the small-world model and centrality measure on the performance.

Window size: Ideally, the window size should be adapted to the logical progression of ideas in the text, and sentences about a particular topic should be grouped into a single window. However, DUC 2002 documents are not divided into logical units such as paragraphs so we define the window size as a fixed number of sentences. A window size of one sentence is too small to capture topics but the window size should remain small relative to the number of sentences in the document to be able to capture the progression of ideas and topics. For large documents, such as the State of the Unions corpus [15] in which most speeches are hundreds of sentences long, a window size of 4 has shown good performance [23]. For short news articles of at most a few tens of sentences (not to mention emails, instant messaging and tweets, which tend to be even shorter), we can expect the information to be much more compact and decreasing the window size might be preferable. As shown in Table 6.3, the optimal window size for the DUC 2002 dataset is again 4 sentences, which therefore looks like a good default value for any small or medium input sizes. However, our experiments showed that the impact of the window size is not very big. This is rather good news as it suggests that choosing the optimal window size is not crucial for obtaining good results.

Target number of edges: Next we investigate the target number of edges. As already pointed out, small-world graphs are typically large graphs; however, the news articles are at most a few tens of sentences long. The concept of small-world is therefore ill-defined with such small graphs. For example, 10-sentence documents are common in the DUC 2002 dataset; in this case, the target number of edges (let us choose, say, $3n$; see page 82 for details) is 30 and a complete graph with 10 vertices contains 45 edges. Hence our graphs are not sparse anymore, unlike small-world graphs. This is why we need to investigate the effect of the target number of edges on small documents and the applicability of the concept of small-world altogether. Optimal results are obtained when no limitation on the number of edges is set (denoted by ∞ in Table 6.3); said differently, it is best to use all keywords from $MeaningfulSet(D, \epsilon = 0)$ when building the graph with short documents. Moreover, Table 6.3 shows that the performance consistently increases as the number of keywords used increases. The reason is probably that small documents do not provide enough data to robustly perform the statistical analysis done by our keywords extraction algorithm. These additional words seem to provide more information that can be exploited successfully by our summarization algorithm.

Small-world models: Regarding the newly introduced small-world models, Table 6.3 shows that the best model on the DUC 2002 dataset is the UB (Undirected Binary) model, which corresponds to the original and simplest model. The UC (Undirected Counts) model also gives good results. The other three models assigning more weight to the words with high meaningfulness values perform significantly worse. This is somewhat surprising and we do not see obvious explanations for this behaviour, but these models might still prove effective with longer documents.

Centrality measure: We now study the performance of several centrality measures. We also investigate the simple ranking function defined as the number of meaningful words inside each sentence. First, despite the fact that the different centrality measures are quite strongly correlated [131], the various measures have a relatively high impact on the performance of the algorithm. The best results are obtained with degree centrality, though closeness centrality gives relatively similar results (see Table 6.3). The simple ranking function also performs quite well, followed by the eigenvector centrality. The betweenness centrality give significantly worse results. The fact that degree centrality outperforms the simple ranking function indicates that benefits can be obtained even from "degenerated" small worlds. Degree centrality also has a large advantage over more indirect centrality measures: its low computational cost.

Discussion

First, it is worth mentioning that our main goal in this section is to study the applicability of our algorithms to small documents and the effect of their parameters. As opposed to many systems in the original competition, we did not try to tune our algorithms for this specific dataset as we did not want to introduce any kind of bias into our system. Indeed, the simple baseline performed very well, which is why most systems in the original DUC 2002 competition were using the sentence index as a feature. This makes sense for news articles as the first sentences usually introduce the main facts and the remaining develop those facts. However, we do not think that such a feature is a good indicator of the importance on a sentence if we do not make any assumptions on the dataset. For example, the first few sentences of a novel do not summarize the remainder of the book.

Regarding processing times, our unoptimized implementation in Python ran in 16.4 seconds for the entire DUC 2002 dataset on an Intel Core 2 E8400 processor (3.0 GHz) with 3 GB of RAM. This represents an average of 28 ms per document, including file opening and parsing, preprocessing, keywords and sentence extraction. The proposed technique allows fast processing of large amounts of data, and thus can be easily adapted to any type of data such as emails and log-files; so it can be used for many real-time applications.

So despite our experimental results being slightly lower than the ones of state-of-the-art summarizers for short documents, our method is both computationally very cheap and language independent. Most importantly, we also showed that small-world modeling, despite being somewhat ill-defined on short documents, remains effective for summarization with appropriate sets of parameters.

6.4.2 Experiments with longer documents

Next, we would like to evaluate our algorithms on longer documents. Indeed, summarization is probably most useful with more substantial documents, where summarization can be utilized to (inter alia):

- help users significantly reduce the time it would take to look through a document in order to decide whether it is worth reading it more carefully,
- build a summary that is comprehensive enough to make reading unnecessary,
- significantly speed-up computations, when used as a preprocessing step.

However, there is no evaluation dataset for long documents, to the best of our knowledge. Indeed, the need for reference summaries would make the creation of such a corpus very expensive. Furthermore, the number of possible summaries increases exponentially as the document size grows and there would probably be little overlap between the different summaries produced by experts. Most of the workshops and research since DUC 2002 have instead been directed towards query-focused and/or multi-document summarization, as in the Text Analysis Conferences (TAC) competitions [11] that replaced DUC in 2008.

Some efforts were made to circumvent some of the problems making summarization evaluation difficult. For example, methods that do not require the use of human models, but rather directly compare the summaries to the original text, were recently investigated. The Jensen-Shannon divergence was shown to have relatively good correlations with pyramid scores and ROUGE-N measures when averaged over many documents [80]. However, none of the studied measures provide a reliable indication of quality for individual documents and those metrics have not been tested on longer documents, so their effectiveness in this setting is unknown.

This highlights, once again, the need for extrinsic evaluation in practical systems. However, since our objective is to build a generic algorithm and in order to show the effectiveness of our algorithm on long documents, we have little choice but to leave the reader judge for himself whether the summaries give a good idea of the contents of the original documents. This also means that comparative evaluation is not possible.

A few examples of summaries from the State of the Union corpus are presented below. The State of the Union speech is an annual address from the President of the United States of America to the US Congress. To make judging easy, only major historic years are used:

- State of the Union by President Truman, during the final months of World War II, and
- State of the Union by President Obama, during the recession following the 2008 subprime's crisis.

The original documents are, respectively, 118 and 291 sentences long.

In these experiments, summary sizes are set to 5 sentences. A target number of edges of $3n$ is used and no stop-words filtering is performed. All small-world models and centrality measures are tested, the window size is varied between 3 and 5, and the best summary is somewhat subjectively chosen.

- **President Truman, April 1945 - Best summary**
  Parameters: Counts Undirected, degree centrality, window size of 5 sentences.

> At a time like this, words are inadequate.
>
> Yet, in this decisive hour, when world events are moving so rapidly, our silence might be misunderstood and might give comfort to our enemies.
>
> In bitter despair, some people have come to believe that wars are inevitable.
>
> During the dark hours of this horrible war, entire nations were kept going by something intangible - hope!
>
> If wars in the future are to be prevented the nations must be united in their determination to keep the peace under law.

- **President Obama, February 2009 - Best summary**
  Parameters: Binary Undirected, closeness centrality, window size of 5 sentences.

> That is what my economic agenda is designed to do, and that is what I'd like to talk to you about tonight.
>
> I want to speak plainly and candidly about this issue tonight, because every American should know that it directly affects you and your family's well-being.
>
> It's a plan that won't help speculators or that neighbor down the street who bought a house he could never hope to afford, but it will help millions of Americans who are struggling with declining home values; Americans who will now be able to take advantage of the lower interest rates that this plan has already helped to bring about.
>
> It's not about helping banks; it's about helping people.
>
> [Applause] It's not about helping banks; it's about helping people.

Overall, these experiments agree in most part with the results from the previous section: the two models "Undirected Binary" and "Undirected Counts" (respectively UB and UC in Table 6.3), together with the degree and closeness centrality measures, provide the best results. A window size of 5 sentences seemed to be slightly superior over window sizes of 3 and 4 sentences. We also generally found the summaries to be quite similar with the various models. However the betweenness and eigenvector centralities seemed to provide relatively different, and seemingly worse, summaries. It may be that more direct definitions of centralities are to prefer for both short and long input documents. For example, changing the closeness centrality for the betweenness centrality with the last example yields:

- **President Obama, February 2009 - Visibly worse summary**
  Parameters: Binary Undirected, eigenvector centrality, window size of 5 sentences

> That is what my economic agenda is designed to do, and that is what I'd like to talk to you about tonight.
>
> Because of this plan, 95 percent of working households in America will receive a tax cut; a tax cut that you will see in your paychecks beginning on April 1st.
>
> I want to speak plainly and candidly about this issue tonight, because every American should know that it directly affects you and your family's well-being.
>
> The ability to get a loan is how you finance the purchase of everything from a home to a car to a college education, how stores stock their shelves, farms buy equipment, and businesses make payroll.
>
> Now, let me be clear–let me be absolutely clear, because I know you'll end up hearing some of the same claims that rolling back these tax breaks means a massive tax increase on the American people: if your family earns less than $\$250,000$ a year, a quarter million dollars a year, you will not see your taxes increased a single dime.

In this summary, tax breaks are emphasized but the economic crisis of 2008 and its roots are less apparent. Finally, let us conclude this section with the summary created by the *First* baseline. The result is a very concrete illustration of its shortcomings:

- **President Obama, February 2009 - *First* baseline**

> Madam Speaker, Mr. Vice President, Members of Congress, the First Lady of the United States – she's around here somewhere: I have come here tonight not only to address the distinguished men and women in this great Chamber, but to speak frankly and directly to the men and women who sent us here.
>
> I know that for many Americans watching right now, the state of our economy is a concern that rises above all others, and rightly so.
>
> If you haven't been personally affected by this recession, you probably know someone who has: a friend, a neighbor, a member of your family.

You don't need to hear another list of statistics to know that our economy is in crisis, because you live it every day.

It's the worry you wake up with and the source of sleepless nights.

6.5 Conclusion

Summarization of textual documents and messages is an important problem in many text mining systems, with applications ranging from quick relevance assessment to speeding up subsequent mining algorithms. Real-life input documents will be varied in genres and lengths (emails, IM conversations, news articles, business reports, etc.), so fast, generic and robust summarization algorithms are needed.

In this chapter, we extended a summarization algorithm that was presented in [24, 25]. Relying on affiliation networks, we proposed several ways to introduce weights to small-world models for textual documents.

Because the algorithms were originally designed for large documents, we demonstrated that, through extensive evaluation, they can be adapted to much shorter inputs and still give good performance. More precisely, we showed that stop-words are no longer automatically removed by the keywords extraction algorithm. Using a standard list of stop-words improves performance; if no such list is available, we introduced an algorithm to automatically generate one (a list of non meaningful words to be precise) from a corpus of documents. We have also shown that the choice of centrality measure is important. Degree centrality performs much better for small documents than other centrality measures. Also, the window size does not have a crucial impact and a size of 4 sentences seems optimal for most inputs. Finally, increasing the target number of edges in the graph was also shown to improve the results.

We also attempted to tune our algorithms for longer inputs by manually checking the resulting summaries. This subjective evaluation showed that degree and closeness centralities give the best results, confirming our previous observations. The graph models didn't seem to have a large impact on the results, so simpler models are preferred. Finally, no stop-words filtering was necessary in this case.

Overall, our models provide a number of ways to efficiently create extractive summaries. Drawing definite conclusions on which model performs best was not possible due to evaluation difficulties. But once again, the solution may lie in extrinsic evaluation, that is, taking into account the specific context for using a summarization algorithm as part of a text mining system.

Chapter 7

Conclusion

Given the complexity of natural language, the task of building text mining systems is difficult. In fact, many natural language processing and understanding problems will require strong AI to be fully solved. In other words, they will require significant progress in artificial intelligence; for example, having a complete and accurate computerized representation of the world will be required to allow advanced reasoning and disambiguation. In the meantime, much simpler approaches have been proposed; among these, the bag-of-words model remains widely popular for many tasks. Ignoring the order of words may seem naive but it has proven time and time again to be extremely effective in practice. Furthermore, thanks to the simplicity of the model, the resulting algorithms can remain computationally cheap.

Numerous ways to derive features from bags-of-words have been proposed over the past several decades, but a recent approach to keywords extraction based on the Helmholtz principle [21, 22] proved appealing for its solid mathematical foundation but also for its ability to distinguish important words from all other words in a parameter-free fashion. In this thesis, we relied on this new approach in order to build novel text mining algorithms which aim at being robust, fast and scalable, as well as theoretically sound.

The resulting algorithms rely on a straightforward preprocessing step that most of the time only consists of word tokenization and stemming. This simplicity allows our algorithms to be readily applicable to many different domains and even languages, as long as a tokenizer and a stemmer are available. Also, by avoiding the reliance on any high complexity steps, our algorithms remain computationally efficient.

More precisely, a new term weighting scheme for information retrieval was proposed in Chapter 4. Its competitiveness with the popular TF-IDF and BM25 schemes has been established and a full system for enterprise search was designed around it. Since current information retrieval algorithms are already robust and fast, the main achievement in this chapter was to establish a theoretically sound and somewhat simpler term weighting scheme that is on par (on the TREC datasets for enterprise search) with current state-of-the-art schemes that are the results of decades of research and fine tuning.

In Chapter 5, a novel algorithm for text segmentation was proposed. Standard evaluation techniques showed its excellent accuracy, especially when stop-words are not removed, as well as a low computational cost that makes it applicable to large documents. As such, it improves on the state-of-the-art algorithms by being faster but also more robust to different genres, domains and languages. This is particularly impressive since speed and robustness were shown to be mutually exclusive with the algorithms evaluated.

Finally in Chapter 6, we generalized and extended the summarization algorithm from [24]. In particular, we relied on the generative model of affiliation networks to propose several ways to introduce weights in textual networks and thus to model documents. Through extensive evaluation, we showed that the algorithm provides good performance on a variety of inputs when used with appropriate sets of parameters. Because it is entirely unsupervised and relies only on straightforward preprocessing steps, our algorithm is relatively generic; and even though it was not explicitly studied in this work, the algorithm can be trivially extended to many western languages in a way similar to our segmentation algorithm. Finally, we also showed that the algorithm is fast.

One can note that the measure of meaningfulness can be adapted to be used as input for a number of other applications; we leave this for future work. As this thesis was being written, we have started working on document categorization and clustering applications. In any case, it is our hope that this research will encourage the development of similarly rigorous approaches to text mining.

Even though little emphasis has been given to software development in this thesis, significant efforts have been directed towards the creation of programs for our industrial partner HP; some of these are presented in the appendices. Thanks to cross functional collaboration with other researchers, program managers, graphic designers and software engineers, the work presented in this thesis has also lead to several successful internal and external pilots.

Appendix A

Demos and pilot projects

A.1 Enterprise Search

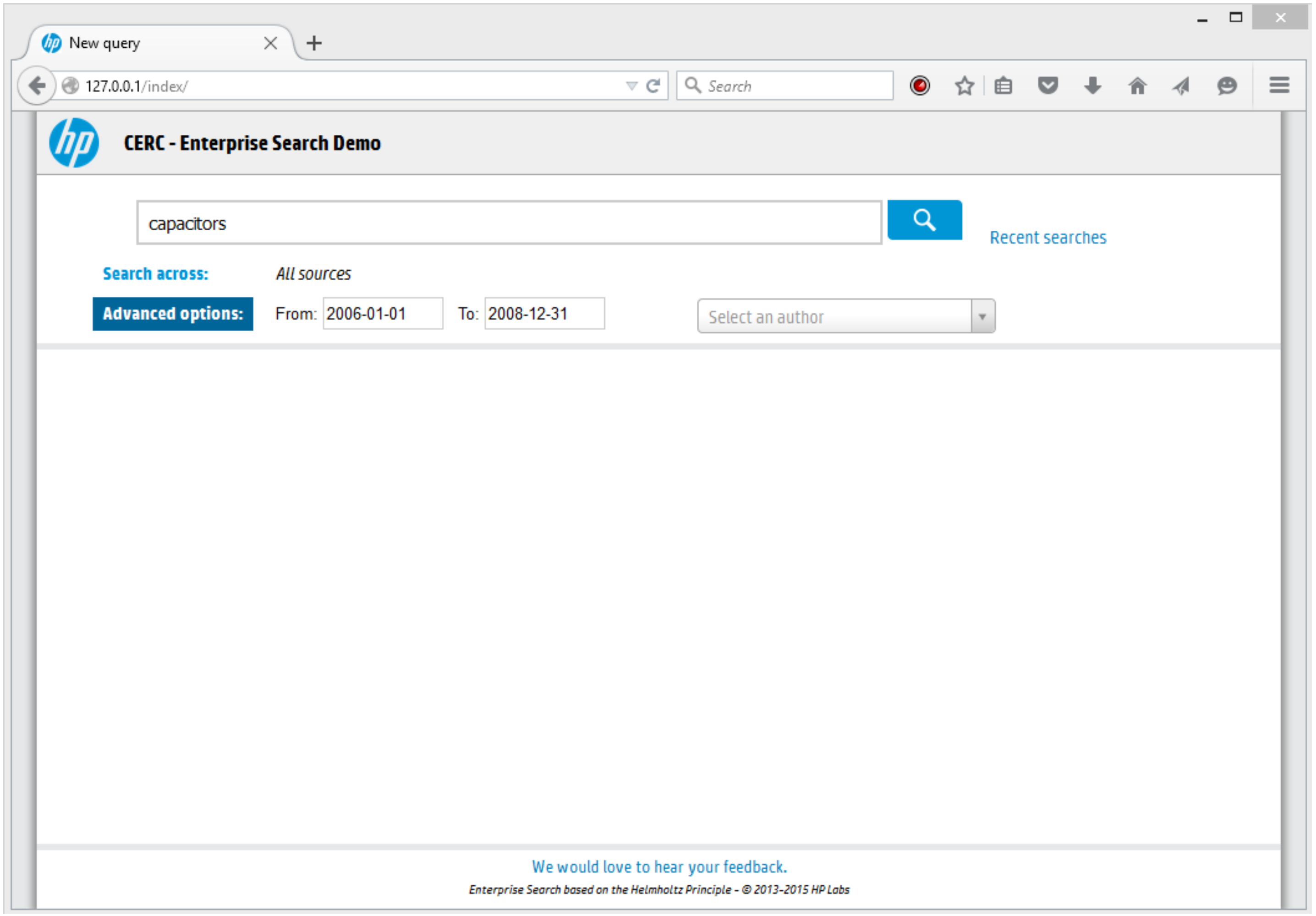
The outcome of our information retrieval research, described in Chapter 4, is a full search engine for enterprise search. A pilot has been led with a large business unit inside HP and several other units have since shown interest.

Figure A.1, page 107, shows the landing page of the search engine, which access can be optionally restricted by a login step. The user is invited to type a query and click the "Search" button. Advanced search functionalities allow to:

- restrict the document sources when several document repositories are indexed,
- search for documents that were created and/or modified in specific date ranges, and
- filter for particular authors.

Figure A.2, page 108, shows examples of search results. Related terms suggestions are displayed to help users refine their query if necessary. Relevant results can be previewed individually or in bulk.

Figure A.3, page 109, shows a document preview. A navigation bar at the top allows to navigate through the results and the user can leave relevance feedback, as well as rate and add comments to documents. This feedback can then be used to improve the search results with learning-to-rank.



CERC - Enterprise Search Demo

capacitors

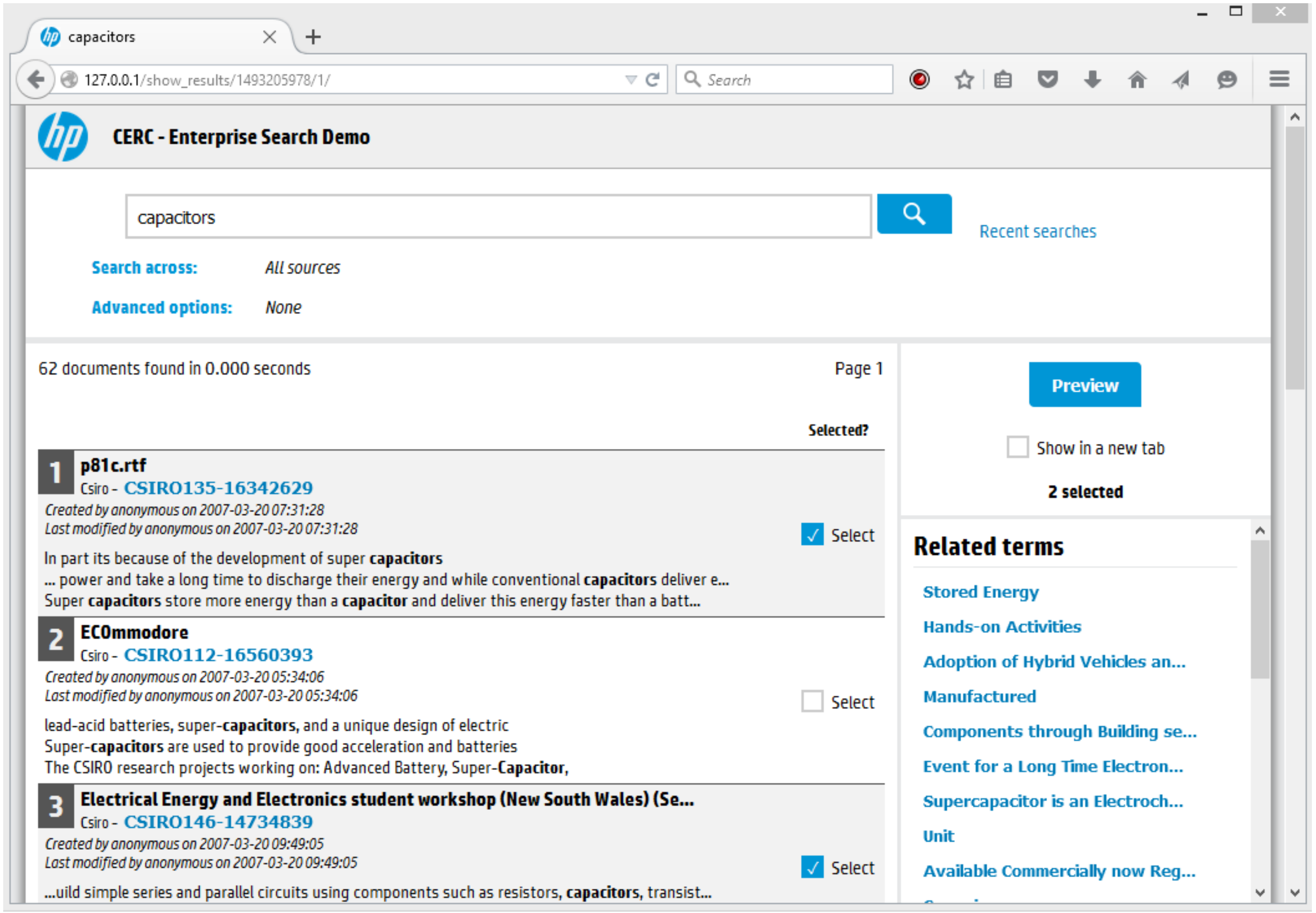
Search across: *All sources*

Advanced options: From: 2006-01-01 To: 2008-12-31 Select an author

We would love to hear your feedback.

Enterprise Search based on the Helmholtz Principle - © 2013-2015 HP Labs

Figure A.1: Search engine landing page.



CERC - Enterprise Search Demo

capacitors

Search across: *All sources*

Advanced options: *None*

62 documents found in 0.000 seconds

Page 1

1 **p81c.rtf**
Csiro - CSIRO135-16342629
Created by anonymous on 2007-03-20 07:31:28
Last modified by anonymous on 2007-03-20 07:31:28
[✓ Select]
In part its because of the development of super **capacitors**
... power and take a long time to discharge their energy and while conventional **capacitors** deliver e...
Super **capacitors** store more energy than a **capacitor** and deliver this energy faster than a batt...

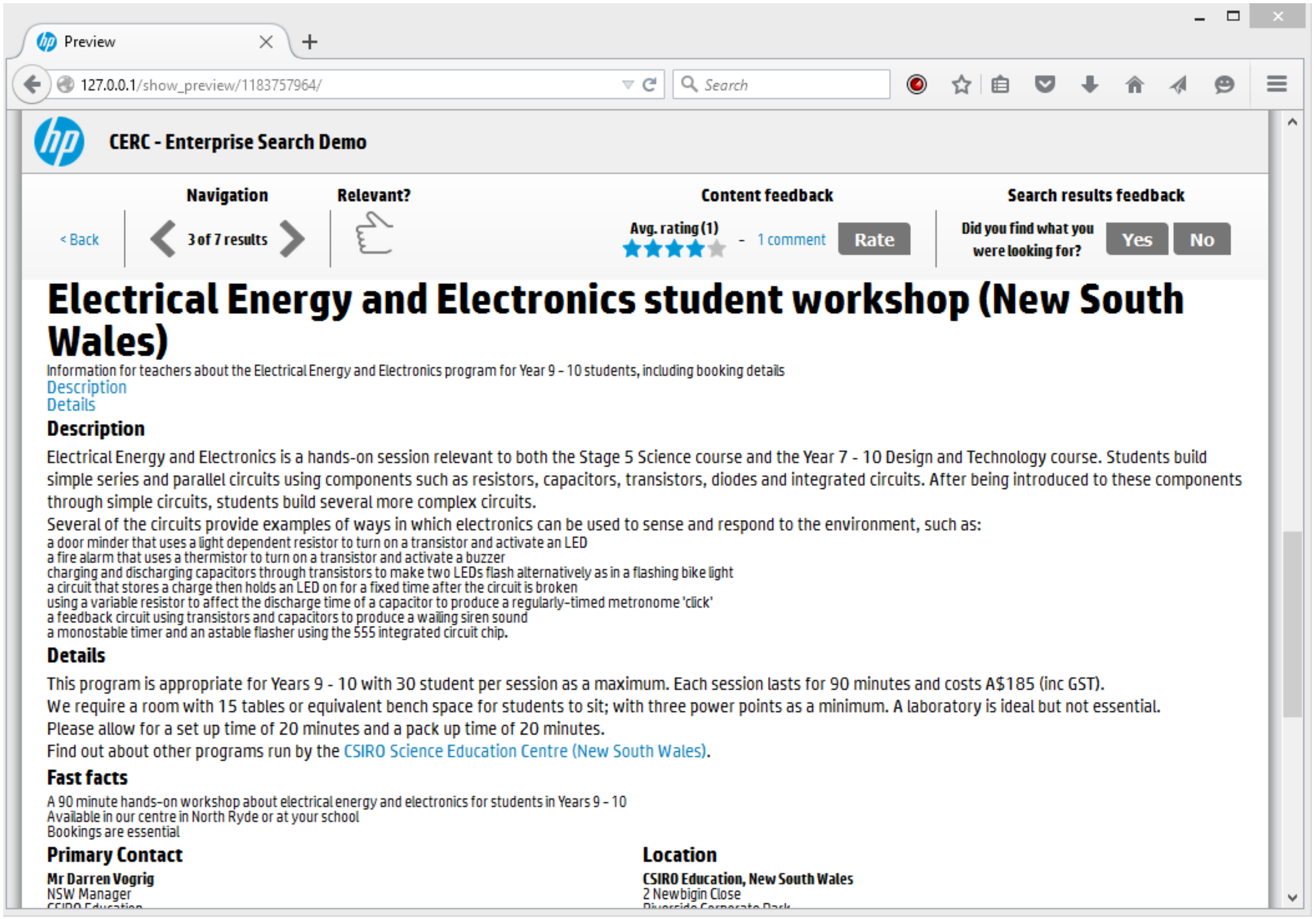
2 **ECOmmodore**
Csiro - CSIRO112-16560393
Created by anonymous on 2007-03-20 05:34:06
Last modified by anonymous on 2007-03-20 05:34:06
Select
lead-acid batteries, super-**capacitors**, and a unique design of electric
Super-**capacitors** are used to provide good acceleration and batteries
The CSIRO research projects working on: Advanced Battery, Super-**Capacitor**,

3 **Electrical Energy and Electronics student workshop (New South Wales) (Se...**
Csiro - CSIRO146-14734839
Created by anonymous on 2007-03-20 09:49:05
Last modified by anonymous on 2007-03-20 09:49:05
[✓ Select]
...uild simple series and parallel circuits using components such as resistors, **capacitors**, transist...

Related terms

- Stored Energy
- Hands-on Activities
- Adoption of Hybrid Vehicles an...
- Manufactured
- Components through Building se...
- Event for a Long Time Electron...
- Supercapacitor is an Electroch...
- Unit
- Available Commercially now Reg...

Figure A.2: Search results for the query "capacitors".



Electrical Energy and Electronics student workshop (New South Wales)

Information for teachers about the Electrical Energy and Electronics program for Year 9 - 10 students, including booking details

Description
Details

Description

Electrical Energy and Electronics is a hands-on session relevant to both the Stage 5 Science course and the Year 7 - 10 Design and Technology course. Students build simple series and parallel circuits using components such as resistors, capacitors, transistors, diodes and integrated circuits. After being introduced to these components through simple circuits, students build several more complex circuits.

Several of the circuits provide examples of ways in which electronics can be used to sense and respond to the environment, such as:

a door minder that uses a light dependent resistor to turn on a transistor and activate an LED
a fire alarm that uses a thermistor to turn on a transistor and activate a buzzer
charging and discharging capacitors through transistors to make two LEDs flash alternatively as in a flashing bike light
a circuit that stores a charge then holds an LED on for a fixed time after the circuit is broken
using a variable resistor to affect the discharge time of a capacitor to produce a regularly-timed metronome 'click'
a feedback circuit using transistors and capacitors to produce a wailing siren sound
a monostable timer and an astable flasher using the 555 integrated circuit chip.

Details

This program is appropriate for Years 9 - 10 with 30 student per session as a maximum. Each session lasts for 90 minutes and costs A$185 (inc GST).

We require a room with 15 tables or equivalent bench space for students to sit; with three power points as a minimum. A laboratory is ideal but not essential.

Please allow for a set up time of 20 minutes and a pack up time of 20 minutes.

Find out about other programs run by the CSIRO Science Education Centre (New South Wales).

Fast facts

A 90 minute hands-on workshop about electrical energy and electronics for students in Years 9 - 10
Available in our centre in North Ryde or at your school
Bookings are essential

Primary Contact

Mr Darren Vogrig
NSW Manager

Location

CSIRO Education, New South Wales
2 Newbigin Close

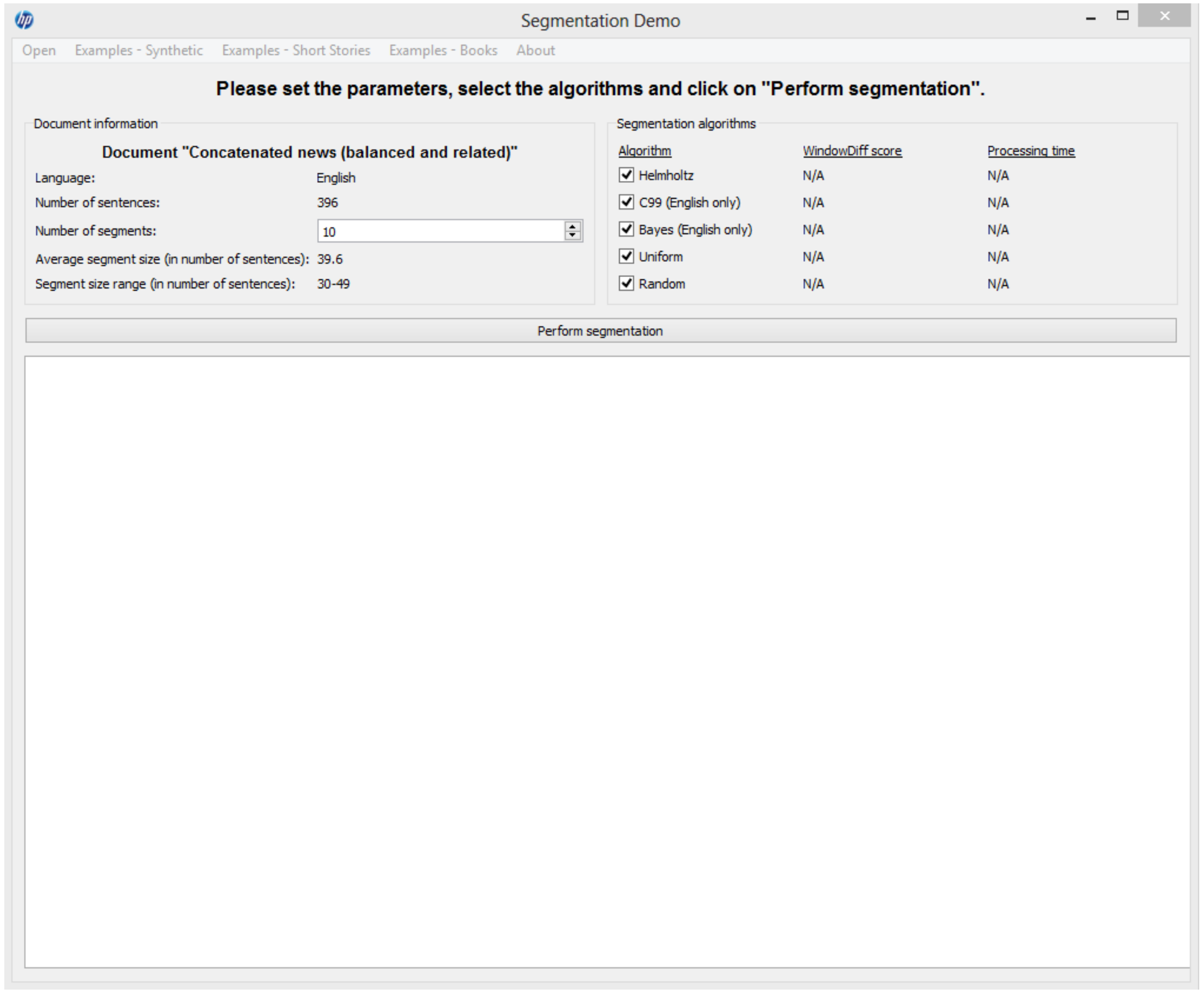
Figure A.3: Previewing a relevant document.

A.2 Segmentation

A segmentation library has been written as part of the research described in Chapter 5. A graphical user interface has been created around it for demonstration purposes. Let us present it.

The user can open a text document or choose one from a large pre-selection of documents of different lengths, genres and languages. Once a document is opened, information and statistics on the input document are displayed, as shown in Figure A.4, page 111. The user can then choose a desired number of segments, select from several state-of-the-art segmentation algorithms and start the segmentation process.

Once the segmentation algorithms have finished to run, each segment found by the algorithm is shown with a different color, as can be seen in Figure A.5, page 112. If a reference segmentation is available, it is also shown using whitespace. On the image, the algorithm has slightly misplaced the true boundary, off by one sentence.



Please set the parameters, select the algorithms and click on "Perform segmentation".

Document information

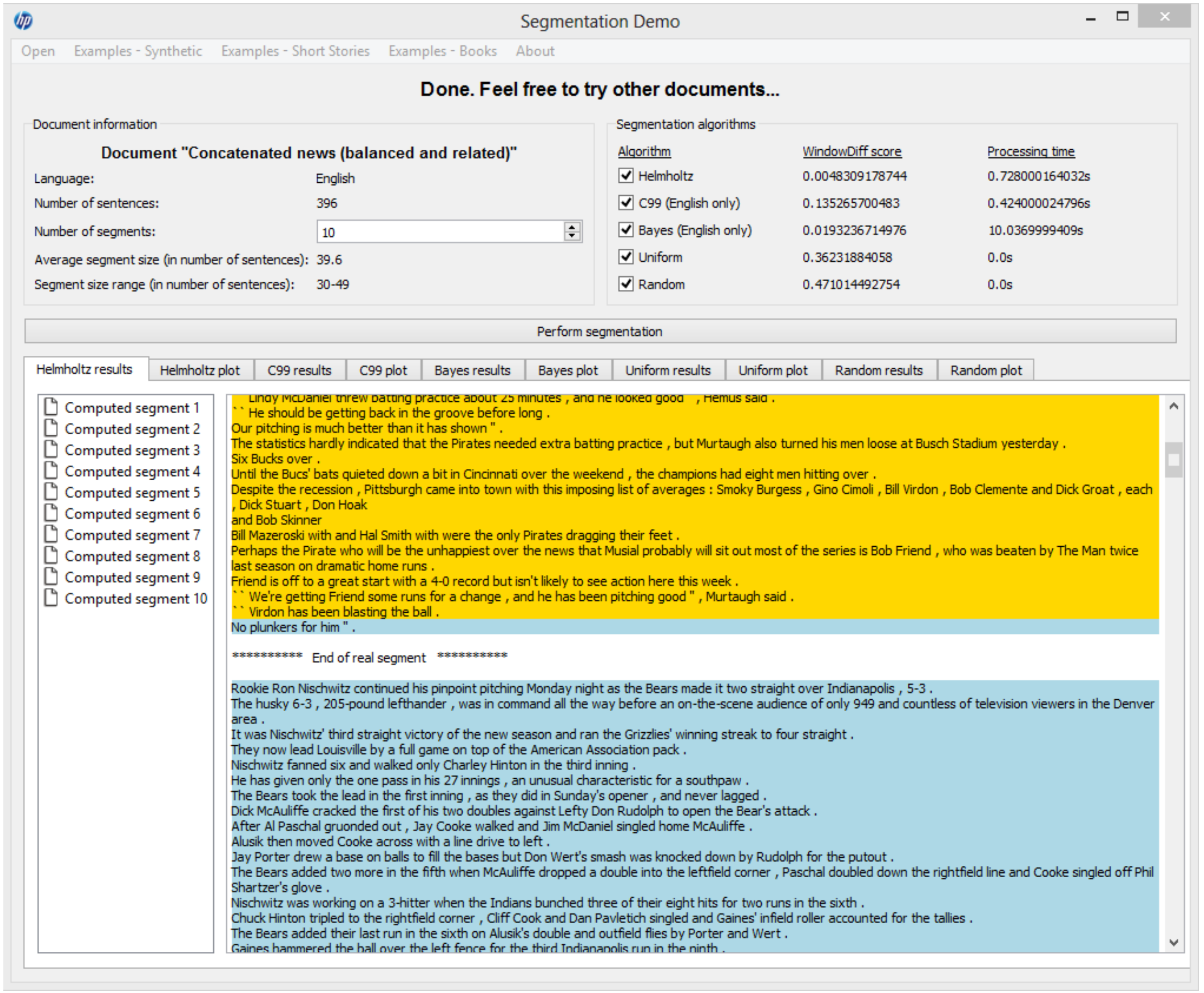
Document "Concatenated news (balanced and related)"

Language:	English
Number of sentences:	396
Number of segments:	10
Average segment size (in number of sentences):	39.6
Segment size range (in number of sentences):	30-49

Segmentation algorithms

Algorithm	WindowDiff score	Processing time
✓ Helmholtz	N/A	N/A
✓ C99 (English only)	N/A	N/A
✓ Bayes (English only)	N/A	N/A
✓ Uniform	N/A	N/A
✓ Random	N/A	N/A

Figure A.4: Segmentation GUI and parameter selection.



Done. Feel free to try other documents...

Document information

Document "Concatenated news (balanced and related)"

Language:	English
Number of sentences:	396
Number of segments:	10
Average segment size (in number of sentences):	39.6
Segment size range (in number of sentences):	30-49

Segmentation algorithms

Algorithm	WindowDiff score	Processing time
✓ Helmholtz	0.0048309178744	0.728000164032s
✓ C99 (English only)	0.135265700483	0.424000024796s
✓ Bayes (English only)	0.0193236714976	10.0369999409s
✓ Uniform	0.36231884058	0.0s
✓ Random	0.471014492754	0.0s

Computed segment 1
Computed segment 2
Computed segment 3
Computed segment 4
Computed segment 5
Computed segment 6
Computed segment 7
Computed segment 8
Computed segment 9
Computed segment 10

```
`` Lindy McDaniel threw batting practice about 25 minutes , and he looked good , Hemus said .
`` He should be getting back in the groove before long .
Our pitching is much better than it has shown " .
The statistics hardly indicated that the Pirates needed extra batting practice , but Murtaugh also turned his men loose at Busch Stadium yesterday .
Six Bucks over .
Until the Bucs' bats quieted down a bit in Cincinnati over the weekend , the champions had eight men hitting over .
Despite the recession , Pittsburgh came into town with this imposing list of averages : Smoky Burgess , Gino Cimoli , Bill Virdon , Bob Clemente and Dick Groat , each
, Dick Stuart , Don Hoak
and Bob Skinner
Bill Mazeroski with and Hal Smith with were the only Pirates dragging their feet .
Perhaps the Pirate who will be the unhappiest over the news that Musial probably will sit out most of the series is Bob Friend , who was beaten by The Man twice
last season on dramatic home runs .
Friend is off to a great start with a 4-0 record but isn't likely to see action here this week .
`` We're getting Friend some runs for a change , and he has been pitching good " , Murtaugh said .
`` Virdon has been blasting the ball .
No plunkers for him " .

**********  End of real segment  **********

Rookie Ron Nischwitz continued his pinpoint pitching Monday night as the Bears made it two straight over Indianapolis , 5-3 .
The husky 6-3 , 205-pound lefthander , was in command all the way before an on-the-scene audience of only 949 and countless of television viewers in the Denver
area .
It was Nischwitz' third straight victory of the new season and ran the Grizzlies' winning streak to four straight .
They now lead Louisville by a full game on top of the American Association pack .
Nischwitz fanned six and walked only Charley Hinton in the third inning .
He has given only the one pass in his 27 innings , an unusual characteristic for a southpaw .
The Bears took the lead in the first inning , as they did in Sunday's opener , and never lagged .
Dick McAuliffe cracked the first of his two doubles against Lefty Don Rudolph to open the Bear's attack .
After Al Paschal gruonded out , Jay Cooke walked and Jim McDaniel singled home McAuliffe .
Alusik then moved Cooke across with a line drive to left .
Jay Porter drew a base on balls to fill the bases but Don Wert's smash was knocked down by Rudolph for the putout .
The Bears added two more in the fifth when McAuliffe dropped a double into the leftfield corner , Paschal doubled down the rightfield line and Cooke singled off Phil
Shartzer's glove .
Nischwitz was working on a 3-hitter when the Indians bunched three of their eight hits for two runs in the sixth .
Chuck Hinton tripled to the rightfield corner , Cliff Cook and Dan Pavletich singled and Gaines' infield roller accounted for the tallies .
The Bears added their last run in the sixth on Alusik's double and outfield flies by Porter and Wert .
Gaines hammered the ball over the left fence for the third Indianapolis run in the ninth .
```

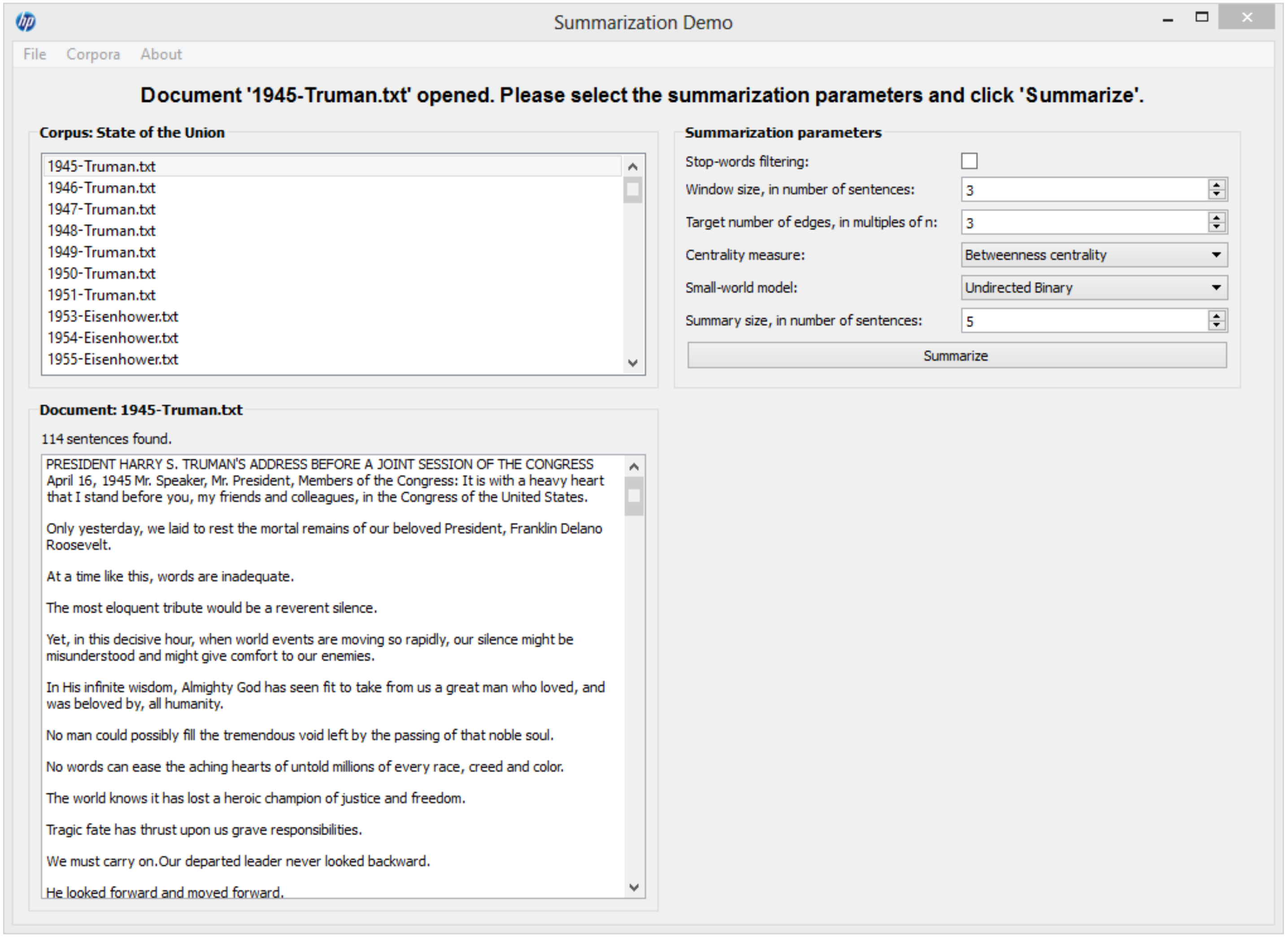
Figure A.5: Segmentation results.

A.3 Summarization

A summarization library has been developed as part of the research described in Chapter 6. This library has been used for on-demand summarization, as part of a web-service available for HP employees. A stand-alone demonstration program, that we briefly describe here, has also been built.

After starting the program, the user can open a text document or choose one from a number of public corpora. The full document is then displayed, as shown in Figure A.6, page 114. After that, the user can choose the summarization parameters such as the window size, the small-world model and the centrality measure (see Chapter 6 for details).

Once the summarization algorithm has been run, the summary is displayed in place of the full document, as shown in Figure A.7, page 115. The small-world graph and associated statistics are also available to make the algorithm easy to visualize and understand.



Document '1945-Truman.txt' opened. Please select the summarization parameters and click 'Summarize'.

Corpus: State of the Union

1945-Truman.txt
1946-Truman.txt
1947-Truman.txt
1948-Truman.txt
1949-Truman.txt
1950-Truman.txt
1951-Truman.txt
1953-Eisenhower.txt
1954-Eisenhower.txt
1955-Eisenhower.txt

Summarization parameters

Stop-words filtering:	☐
Window size, in number of sentences:	3
Target number of edges, in multiples of n:	3
Centrality measure:	Betweenness centrality
Small-world model:	Undirected Binary
Summary size, in number of sentences:	5

Document: 1945-Truman.txt

114 sentences found.

PRESIDENT HARRY S. TRUMAN'S ADDRESS BEFORE A JOINT SESSION OF THE CONGRESS April 16, 1945 Mr. Speaker, Mr. President, Members of the Congress: It is with a heavy heart that I stand before you, my friends and colleagues, in the Congress of the United States.

Only yesterday, we laid to rest the mortal remains of our beloved President, Franklin Delano Roosevelt.

At a time like this, words are inadequate.

The most eloquent tribute would be a reverent silence.

Yet, in this decisive hour, when world events are moving so rapidly, our silence might be misunderstood and might give comfort to our enemies.

In His infinite wisdom, Almighty God has seen fit to take from us a great man who loved, and was beloved by, all humanity.

No man could possibly fill the tremendous void left by the passing of that noble soul.

No words can ease the aching hearts of untold millions of every race, creed and color.

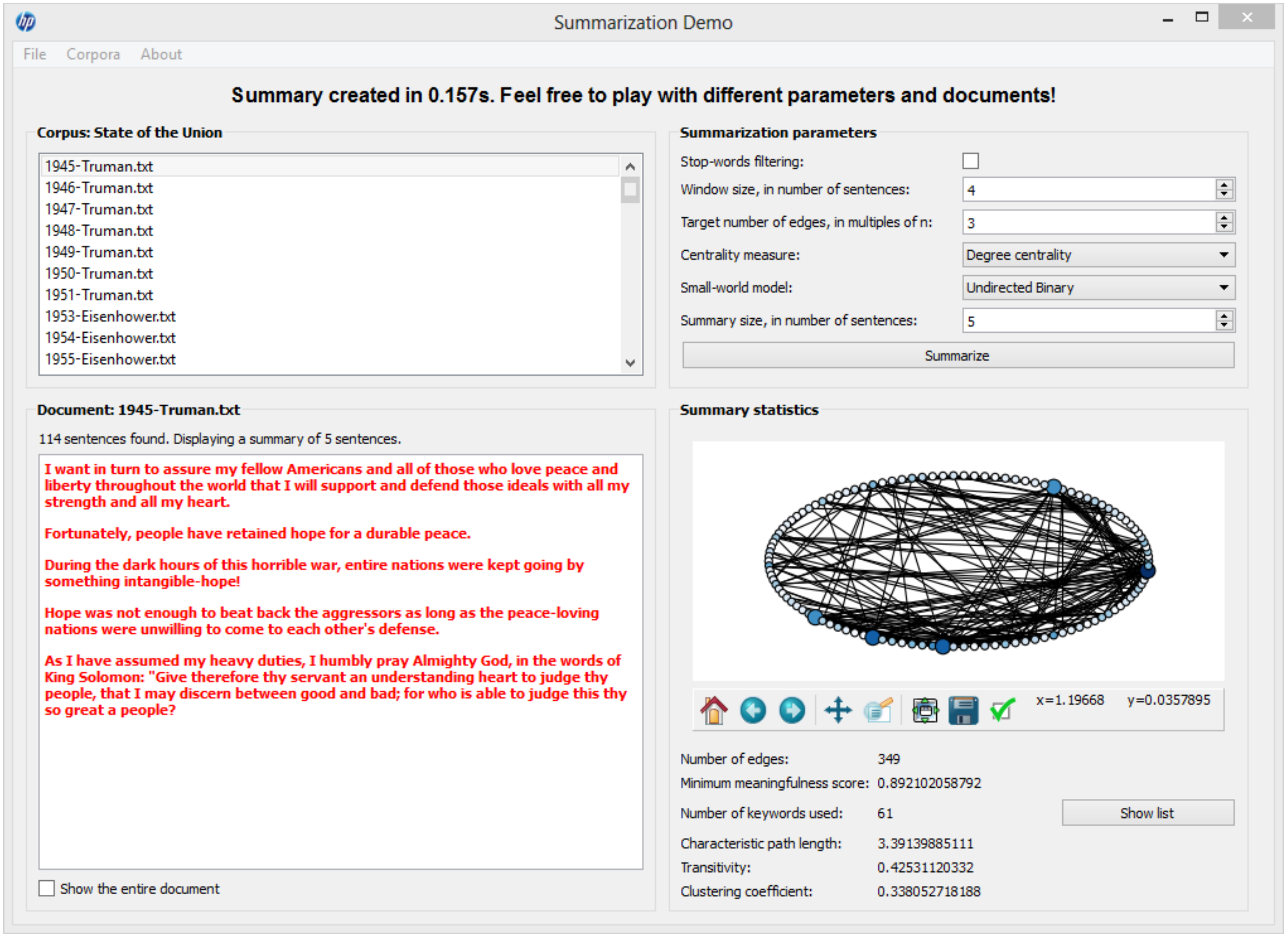
The world knows it has lost a heroic champion of justice and freedom.

Tragic fate has thrust upon us grave responsibilities.

We must carry on.Our departed leader never looked backward.

He looked forward and moved forward.

Figure A.6: Summarization GUI and parameter selection.



Summary created in 0.157s. Feel free to play with different parameters and documents!

Corpus: State of the Union

1945-Truman.txt
1946-Truman.txt
1947-Truman.txt
1948-Truman.txt
1949-Truman.txt
1950-Truman.txt
1951-Truman.txt
1953-Eisenhower.txt
1954-Eisenhower.txt
1955-Eisenhower.txt

Summarization parameters

Stop-words filtering:	☐
Window size, in number of sentences:	4
Target number of edges, in multiples of n:	3
Centrality measure:	Degree centrality
Small-world model:	Undirected Binary
Summary size, in number of sentences:	5

Document: 1945-Truman.txt

114 sentences found. Displaying a summary of 5 sentences.

I want in turn to assure my fellow Americans and all of those who love peace and liberty throughout the world that I will support and defend those ideals with all my strength and all my heart.

Fortunately, people have retained hope for a durable peace.

During the dark hours of this horrible war, entire nations were kept going by something intangible-hope!

Hope was not enough to beat back the aggressors as long as the peace-loving nations were unwilling to come to each other's defense.

As I have assumed my heavy duties, I humbly pray Almighty God, in the words of King Solomon: "Give therefore thy servant an understanding heart to judge thy people, that I may discern between good and bad; for who is able to judge this thy so great a people?

☐ Show the entire document

Summary statistics

Number of edges:	349
Minimum meaningfulness score:	0.892102058792
Number of keywords used:	61
Characteristic path length:	3.39139885111
Transitivity:	0.42531120332
Clustering coefficient:	0.338052718188

Figure A.7: Summarization results.

List of Figures

2.1 Simplified, high-level architecture of a text mining system 3

3.1 Illustrations of the main grouping principles from the Gestalt theory . 15

3.2 Illustration of the balls-and-bins experiment, with K balls thrown uniformly at random into N bins 18

3.3 Using the adaptive number of containers N_{adapt} to cope with containers of varying lengths 21

4.1 Impact of the number of top documents re-ranked on various metrics and on the computational cost 48

5.1 Cosine similarity computation between adjacent blocks of 120 words, in the TextTiling algorithm [61] 51

5.2 Main steps of the C99 algorithm [36] 52

5.3 The graph G for an example document of 5 sentences in *TextSeg* [130] 54

5.4 Segmenting is finding which of the $n - 1$ gaps between sentences best separate the topics inside the documents 56

5.5 Activity stretch of a word 57

5.6 Unsmoothed gap scores and ground truth boundaries 58

5.7 Effect of the parameter σ for smoothing the gap score curve 59

5.8 Smoothed gap scores and final boundaries identified by our algorithm 60

5.9 Comparison of the boundaries from the ground truth segmentation and from the output of our algorithm 60

5.10 Gap score plots of "Alice in Wonderland" and European translations, obtained with our algorithm 74

6.2 Small-world building procedure. 83

6.3 An example of affiliation network and the corresponding social graph 84

A.1 Search engine landing page. 107

A.2 Search results for the query "capacitors". 108

A.3 Previewing a relevant document. 109

A.4 Segmentation GUI and parameter selection. 111

A.5 Segmentation results. 112

A.6 Summarization GUI and parameter selection. 114

A.7 Summarization results. 115

List of Tables

4.1 A simple inverted index 24
4.2 Query-document features. 32
4.3 Document features. 34
4.4 Query features. 35
4.5 Comparison of term weighting formulae for initial ranking 41
4.6 Impact of filtering non-meaningful words on initial rankings. 43
4.7 Impact of the learning-to-rank algorithm on re-ranking, with top 25 documents re-ranked 45
4.8 Impact of the number of top documents on re-ranking, using a pairwise AdaBoost algorithm 47

5.1 Composition of Choi's synthetic dataset [36]. 64
5.2 WindowDiff scores on the synthetic datasets with stop-words removal 65
5.3 WindowDiff scores on the synthetic datasets without stop-words removal 65
5.4 WindowDiff scores on the synthetic datasets with different stop-words list 66
5.5 Short stories dataset statistics 68
5.6 WindowDiff scores on the short stories dataset 68
5.7 Novels dataset statistics 70
5.8 WindowDiff scores on the novels dataset 70
5.9 Alice in Wonderland dataset statistics 72

5.10 WindowDiff scores on the Alice in Wonderland dataset 72

6.1 Ranking words of a fictitious document D according to their $\max_{P\in\mathcal{P}}$ $meaning(w, P, D)$ value 81

6.2 Results on the DUC 2002 dataset, using ROUGE-1 F-score. 95

6.3 Effect of our algorithm's parameters on the results on the DUC 2002 dataset, using ROUGE-1 F-score 96

List of Algorithms

4.1 Initial ranking computations. 31

5.1 Gap scores calculation. 57

6.1 Original summarization algorithm [24, 25]. 83

6.2 Complete summarization algorithm. 91

Index

ϵ-meaningful event, 16

Affiliation matrix, 81
Affiliation network, 80
  Directed graph, 82
  Undirected graph, 82
Anomaly detection, 10

Bag-of-words, 8

Categorization, 10
Classification, 10
Clustering, 10

Dimensionality reduction, 8

Entity
  extraction, 6
  recognition, 6

Gestalt, 13, 14

Helmholtz principle, 15

Index, 23
  inverted, 23
Information filtering, 10
Information retrieval, 9, 22
  Algebraic models, 24
  Boolean models, 24
  Probabilistic models, 26
  Set-theoretic models, 24
  Vector space models, 24
Inverse Document Frequency (IDF), 25

Keyword, 18

Learning-to-Rank, 26
  listwise approach, 27
  pairwise approach, 27
  pointwise approach, 27

Meaningful event, 16
Meaningful Set, 78

Normalization
  word, 5
Normalized Discounted Cumulative Gain (nDCG), 35
Number of False Alarms (NFA), 16

Parsing
  syntactic, 6
Part-of-speech
  tagging, 6
Perception
  visual, 13
Precision
  Average Precision (AP), 35
Preprocessing, 4

Reciprocal Rank (RR), 35
Retrieval model, 24
ROUGE, 89
  ROUGE-N, 90

Semantic analysis, 11
Small-world, 78
Stemming, 6
Stop-word, 5
  filtering, 5
  removal, 5
Summary, 10
  abstractive, 74
  extractive, 74
  generic, 74
  multi-document, 74
  query-focused, 74
  single-document, 74

Term Frequency (TF), 25
Text Analytics, 3
Text mining, 7
TF-IDF, 24
Time series, 10
Tokenization, 5

Unstructured data, 9

Weighting scheme, 8

Bibliography

[1] CSIRO. http://www.csiro.au. Accessed October 1, 2015.

[2] Document Understanding Conference (DUC). http://duc.nist.gov. Accessed October 1, 2015.

[3] Google Books: N-Gram Viewer. https://books.google.com/ngrams. Accessed October 1, 2015.

[4] Hausdorff distance. https://en.wikipedia.org/wiki/Hausdorff_distance. Accessed October 1, 2015.

[5] List of stop-words. http://www.jmlr.org/papers/volume5/lewis04a/a11-smart-stop-list/english.stop. Accessed October 1, 2015.

[6] Natural Language ToolKit. http://www.nltk.org. Accessed October 1, 2015.

[7] Project Gutenberg: Free eBooks. https://www.gutenberg.org. Accessed October 1, 2015.

[8] RankLib. http://sourceforge.net/p/lemur/wiki/RankLib/. Accessed October 1, 2015.

[9] scikit-learn: Machine Learning in Python. http://scikit-learn.org. Accessed October 1, 2015.

[10] SVMRank: Support Vector Machine for Ranking. http://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html. Accessed October 1, 2015.

[11] Text Analysis Conference (TAC). http://www.nist.gov/tac/. Accessed October 1, 2015.

[12] The Machine: A new kind of computer. http://www.hpl.hp.com/research/systems-research/themachine/. Accessed October 1, 2015.

[13] The Penn Treebank Project, Treebank tokenization. `http://www.cis.upenn.edu/~treebank/tokenization.html`. Accessed October 1, 2015.

[14] UK Government Security Classifications, April 2014. `https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/251480/Government-Security-Classifications-April-2014.pdf`. Accessed October 1, 2015.

[15] Union Addresses from 1709. `http://stateoftheunion.onetwothree.net`. Accessed October 1, 2015.

[16] Wikipedia, the free encyclopedia. `https://www.wikipedia.org`. Accessed October 1, 2015.

[17] Wikisource, the free library. `https://www.wikisource.org`. Accessed October 1, 2015.

[18] G. Amati and C. J. Van Rijsbergen. Probabilistic models of information retrieval based on measuring the divergence from randomness. *ACM Transactions on Information Systems*, 20(4):357–389, 2002.

[19] P. Bailey, N. Craswell, I. Soboroff, and A. P. de Vries. The CSIRO enterprise search test collection. *SIGIR Forum*, 41(2):42–45, 2007.

[20] P. Bailey, A. P. D. Vries, N. Craswell, and I. Soboroff. Overview of the TREC 2007 enterprise track. In *Proceedings of the 14th Text REtrieval Conference (TREC)*, 2007.

[21] A. Balinsky, H. Balinsky, and S. Simske. On the Helmholtz principle for documents processing. In *Proceedings of the 10th ACM Symposium on Document Engineering*, 2010.

[22] A. Balinsky, H. Balinsky, and S. Simske. On the Helmholtz principle for data mining. In *Proceedings of the Conference on Knowledge Discovery*, 2011.

[23] A. Balinsky, H. Balinsky, and S. Simske. Rapid change detection and text mining. In *Proceedings of the 2nd IMA Conference on Mathematics in Defence*, 2011.

[24] H. Balinsky, A. Balinsky, and S. Simske. Automatic text summarization and small-world networks. In *Proceedings of the 11th ACM Symposium on Document Engineering*, 2011.

[25] H. Balinsky, A. Balinsky, and S. Simske. Document sentences as a small world. In *Proceedings of the IEEE Conference on Systems, Man, and Cybernetics*, 2011.

[26] H. Balinsky and B. Dadachev. Automatic Text and Data Stream Segmentation using Principles of Human Perception. In *Proceedings of the HP Imaging and Color Symposium*, 2013.

[27] K. Balog, I. Soboroff, P. Thomas, N. Craswell, A. P. de Vries, and P. Bailey. Overview of the TREC 2008 enterprise track. In *Proceedings of the 17th Text REtrieval Conference Proceedings (TREC)*, 2008.

[28] R. Barzilay and M. Elhadad. Using lexical chains for text summarization. In *Proceedings of the ACL Workshop on Intelligent Scalable Text Summarization*, pages 10–17, 1997.

[29] R. E. Bellman. *Adaptive control processes - A guided tour*. Princeton University Press, 1961.

[30] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, 2006.

[31] D. Blei, A. Ng, and M. Jordan. Latent Dirichlet allocation. *Machine Learning Research*, 3:993–1022, 2003.

[32] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Computer Networks and ISDN Systems*, pages 107–117, 1998.

[33] V. Bruce, P. Green, and M. Georgeson. *Visual Perception: Physiology, Psychology, & Ecology*. Psychology Press, 2003.

[34] A. Carstairs-McCarthy. *An Introduction to English Morphology: Words and Their Structure*. Edinburgh University Press, 2002.

[35] Y. Chali, S. A. Hasan, and S. R. Joty. Do automatic annotation techniques have any impact on supervised complex question answering? In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 329–332, 2009.

[36] F. Choi. Advances in domain independent linear text segmentation. In *Proceedings of the North American Chapter of the Association for Computational Linguistics Conference*, pages 26–33, 2000.

[37] E. Clark and K. Araki. Text normalization in social media: Progress, problems and applications for a pre-processing system of casual English. *Procedia - Social and Behavioral Sciences*, 27(0):2–11, 2011.

[38] M. Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL Conference on Empirical Methods in Natural Language Processing*, pages 1–8, 2002.

[39] C. D. Corley, D. J. Cook, A. R. Mikler, and K. P. Singh. Text and structural data mining of influenza mentions in web and social media. *International journal of environmental research and public health*, 7(2):596–615, 2010.

[40] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[41] B. Dadachev, A. Balinsky, and H. Balinsky. On Automatic Text Segmentation. In *Proceedings of the 14th ACM Symposium on Document Engineering*, 2014.

[42] B. Dadachev, A. Balinsky, H. Balinsky, and G. Forman. Automatic Text and Data Stream Segmentation using Weighted Feature Extraction. In *Proceedings of the 3rd IMA Conference on Mathematics in Defence*, 2013.

[43] B. Dadachev, A. Balinsky, H. Balinsky, and S. Simske. On the Helmholtz Principle for Data Mining. In *Proceedings of the 3rd IEEE Conference on Emerging Security Technologies*, 2012.

[44] E. David and K. Jon. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World.* Cambridge University Press, 2010.

[45] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[46] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[47] A. Desolneux, L. Moisan, and J.-M. Morel. *From Gestalt Theory to Image Analysis: A Probabilistic Approach.* Springer, 2008.

[48] J. Eisenstein and R. Barzilay. Bayesian unsupervised topic segmentation. In *Proceedings of the ACL Conference on Empirical Methods in Natural Language Processing*, pages 334–343, 2008.

[49] G. Erkan and D. R. Radev. LexRank: Graph-based lexical centrality as salience in text summarization. *Journal of Artificial Intelligence Research*, 22(1):457–479, 2004.

[50] L. C. Freeman. Centrality in social networks: conceptual clarification. *Social Networks*, 1:215–239, 1978.

[51] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Science*, 55(1):119–139, 1997.

[52] M. Galley and K. McKeown. Improving word sense disambiguation in lexical chaining. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 1486–1488, 2003.

[53] K. Ganesan, C. Zhai, and J. Han. Opinosis: A graph-based approach to abstractive summarization of highly redundant opinions. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 340–348, 2010.

[54] Y. Gong and X. Liu. Generic text summarization using relevance measure and latent semantic analysis. In *Proceedings of the 24th ACM Conference on Research and Development in Information Retrieval*, pages 19–25, 2001.

[55] M. V. C. Guelpeli, A. Garcia, and A. Branco. The process of summarization in the pre-processing stage in order to improve measurement of texts when clustering. In *Proceedings of the Conference for Internet Technology and Secured Transactions*, pages 388–395, 2011.

[56] Z. Gyöngyi and H. Garcia-Molina. Web spam taxonomy. In *Proceedings of the 1st International Workshop on Adversarial Information Retrieval on the Web*, pages 39–47, 2005.

[57] A. Halevy, P. Norvig, and F. Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2):8–12, 2009.

[58] M. A. Halliday and R. Hasan. *Cohesion in English*. Longman, 1976.

[59] R. A. Hanneman and M. Riddle. *Introduction to social network methods*. 2005.

[60] C. Havasi, R. Speer, and J. Alonso. Conceptnet 3: a flexible, multilingual semantic network for common sense knowledge. In *Recent advances in natural language processing*, pages 27–29, 2007.

[61] M. Hearst. TextTiling: segmenting text into multi-paragraph subtopic passages. *Computational Linguistics*, 23(1):33–64, 1997.

[62] T. Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd ACM Conference on Research and Development in Information Retrieval*, pages 50–57, 1999.

[63] M. Y. Ivory and M. A. Hearst. Statistical profiles of highly-rated web sites. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pages 367–374, 2002.

[64] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 133–142, 2002.

[65] T. Joachims. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 217–226, 2006.

[66] A. Joly and O. Buisson. Logo retrieval with a contrario visual query expansion. In *Proceedings of the 17th ACM International Conference on Multimedia*, pages 581–584, 2009.

[67] F. Katamba. *English Words*. Routledge, 1994.

[68] T. Kiss and J. Strunk. Unsupervised multilingual sentence boundary detection. *Computational Linguistics*, 32(4):485–525, 2006.

[69] D. Klein and C. D. Manning. Fast exact inference with a factored model for natural language parsing. *Advances in Neural Information Processing Systems*, 15:3–10, 2003.

[70] J. Kleinberg. Bursty and hierarchical structure in streams. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 91–101, 2002.

[71] J. Kupiec, J. Pedersen, and F. Chen. A trainable document summarizer. In *Proceedings of the 18th ACM Conference on Research and Development in Information Retrieval*, pages 68–73, 1995.

[72] S. Lamprier, T. Amghar, B. Levrat, and F. Saubion. On evaluation methodologies for text segmentation algorithms. In *Proceedings of the Conference on Tools with Artificial Intelligence*, pages 19–26, 2007.

[73] S. Lattanzi and D. Sivakumar. Affiliation networks. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pages 427–434, 2009.

[74] H. Li. A short introduction to learning to rank. *IEICE Transactions*, 94-D(10):1854–1862, 2011.

[75] C.-Y. Lin. ROUGE: a package for automatic evaluation of summaries. In *Proceedings of the Workshop on Text Summarization Branches Out*, 2004.

[76] C.-Y. Lin and E. Hovy. The automated acquisition of topic signatures for text summarization. In *Proceedings of the 18th Conference on Computational Linguistics*, pages 495–501, 2000.

[77] T. Lindeberg. *Scale-Space Theory in Computer Vision*. Springer, 1993.

[78] D. Litman and R. Passonneau. Combining multiple knowledge sources for discourse segmentation. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 108–115, 1995.

[79] T.-Y. Liu, J. Xu, T. Qin, W. Xiong, and H. Li. LETOR: Benchmark dataset for research on learning-to-rank for information retrieval. In *Proceedings of the Workshop on Learning-to-Rank for Information Retrieval*, pages 3–10, 2007.

[80] A. Louis and A. Nenkova. Automatically evaluating content selection in summarization without human models. In *Proceedings of the ACL Conference on Empirical Methods in Natural Language Processing*, pages 306–314, 2009.

[81] D. G. Lowe. *Perceptual Organization and Visual Recognition*. Kluwer Academic Publishers, 1985.

[82] H. P. Luhn. The automatic creation of literature abstracts. *IBM Journal of Research & Development*, 2(2):159–165, 1958.

[83] C. Macdonald, R. L. Santos, and I. Ounis. On the usefulness of query features for learning to rank. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, pages 2559–2562, 2012.

[84] I. Malioutov and R. Barzilay. Minimum cut model for spoken lecture segmentation. In *Proceedings of the 21st International Conference on Computational Linguistics*, pages 25–32, 2006.

[85] I. Mani. Summarization evaluation: An overview. Technical report, The MITRE corporation, 2001.

[86] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. Also available as `http://nlp.stanford.edu/IR-book/pdf/irbookprint.pdf`.

[87] M. Mathioudakis and N. Koudas. TwitterMonitor: trend detection over the twitter stream. In *Proceedings of the ACM International Conference on Management of data*, pages 1155–1158, 2010.

[88] A. McCallum and W. Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proceedings of the 7th Conference on Natural Language Learning*, pages 188–191, 2003.

[89] W. Metzger. *Gesetze des Sehens*. Kramer, 1975.

[90] W. Metzger. Was ist gestalttheorie? *Gestalttheorie und Erziehung*, 508:1–17, 1975.

[91] J.-B. Michel, Y. K. Shen, A. P. Aiden, A. Veres, M. K. Gray, T. G. B. Team, J. P. Pickett, D. Holberg, D. Clancy, P. Norvig, J. Orwant, S. Pinker, M. A. Nowak, and E. L. Aiden. Quantitative analysis of culture using millions of digitized books. *Science*, 331(6014), 2010.

[92] R. Mihalcea and P. Tarau. TextRank: Bringing order into texts. In *Proceedings of the ACL Conference on Empirical Methods in Natural Language Processing*, pages 404–411, 2004.

[93] S. Milgram. The Small World Problem. *Psychology Today*, 2:60–67, 1967.

[94] G. A. Miller. WordNet: A lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[95] J. Morris and G. Hirst. Lexical cohesion computed by thesaural relations as an indicator of the structure of text. *Computational Linguistics*, 17(1):21–48, 1991.

[96] P. Moulon, P. Monasse, and R. Marlet. Adaptive structure from motion with a contrario model estimation. In *Proceedings of the Asian Conference in Computer Vision*, pages 257–270. 2013.

[97] A. Nenkova and K. McKeown. Automatic summarization. *Foundations and Trends in Information Retrieval*, 5(2-3):103–233, 2011.

[98] A. Nenkova and K. McKeown. A survey of text summarization techniques. In *Mining Text Data*, pages 43–76. Springer, 2012.

[99] A. Nenkova and R. J. Passonneau. Evaluating content selection in summarization: The pyramid method. In *Proceedings of the North American Chapter of the Association for Computational Linguistics Conference*, pages 145–152, 2004.

[100] A. Nenkova and L. Vanderwende. The impact of frequency on summarization. Technical report, Microsoft Research, 2005.

[101] M. E. Newman. The mathematics of networks. *The new Palgrave encyclopedia of economics*, 2:1–12, 2008.

[102] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, 1999.

[103] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10), 2010.

[104] L. Pevzner and M. Hearst. A critique and improvement of an evaluation metric for text segmentation. *Computational Linguistics*, 28(1):19–36, 2002.

[105] O. Phelan, K. McCarthy, and B. Smyth. Using twitter to recommend real-time topical news. In *Proceedings of the 3rd ACM Conference on Recommender Systems*, pages 385–388, 2009.

[106] J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st ACM Conference on Research and Development in Information Retrieval*, pages 275–281, 1998.

[107] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[108] M. F. Porter. Snowball: A language for stemming algorithms. 2001. Available at `http://snowball.tartarus.org/texts/introduction.html`.

[109] T. Qin and T.-Y. Liu. Introducing LETOR 4.0 Datasets. *Computing Research Repository (CoRR)*, 2013.

[110] D. R. Radev, H. Jing, M. Styś, and D. Tam. Centroid-based summarization of multiple documents. *Information Processing & Management*, 40(6):919–938, 2004.

[111] S. Robertson. Understanding inverse document frequency: On theoretical arguments for idf. *Journal of Documentation*, 60:2004, 2004.

[112] S. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In *Proceedings of the 3rd Text REtrieval Conference (TREC)*, pages 109–126, 1996.

[113] S. Robertson and H. Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009.

[114] S. Robertson, H. Zaragoza, and M. Taylor. Simple BM25 extension to multiple weighted fields. In *Proceedings of the 13th ACM International Conference on Information and Knowledge Management*, pages 42–49, 2004.

[115] S. E. Robertson and K. Sparck Jones. Relevance weighting of search terms. In *Document Retrieval Systems*, pages 143–160. 1988.

[116] F. B. Rogers. Medical subject headings. *Bulletin of the Medical Library Association*, 51:114–116, 1963.

[117] G. Salton. *The SMART Retrieval System - Experiments in Automatic Document Processing*. Prentice-Hall, 1971.

[118] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513–523, 1988.

[119] G. Salton, E. A. Fox, and H. Wu. Extended Boolean information retrieval. *Communications of the ACM*, 26(11):1022–1036, 1983.

[120] J. D. Schlesinger, M. E. Okurowski, J. M. Conroy, D. P. O'Leary, A. Taylor, J. Hobbs, and H. T. Wilson. Understanding machine performance in the context of human performance for multi-document summarization. In *Proceedings of the Document Understanding Conference*, 2002.

[121] J. Silla, Carlos N. and C. A. Kaestner. An analysis of sentence boundary detection systems for English and Portuguese documents. *Lecture Notes in Computer Science: Computational Linguistics and Intelligent Text Processing*, pages 135–141, 2004.

[122] R. Sproat, A. W. Black, S. F. Chen, S. Kumar, M. Ostendorf, and C. Richards. Normalization of non-standard words. *Computer Speech & Language*, 15(3):287–333, 2001.

[123] M. Steinbach, G. Karypis, V. Kumar, et al. A comparison of document clustering techniques. In *Proceedings of the KDD Workshop on Text Mining*, pages 525–526, 2000.

[124] J. Su, J. S. Shirab, and S. Matwin. Large scale text classification using semi-supervised multinomial naive Bayes. In *Proceedings of the 28th International Conference on Machine Learning*, pages 97–104, 2011.

[125] D. R. Swanson. Information Retrieval as a Trial-and-Error Process. *Library Quarterly*, 47(2):128–148, 1977.

[126] V. Tahani. A conceptual framework for fuzzy query processing - a step toward very intelligent database systems. *Information Processing & Management*, 13(5):289–303, 1977.

[127] T. Tao and C. Zhai. An exploration of proximity measures in information retrieval. In *Proceedings of the 30th ACM Conference on Research and Development in Information Retrieval*, pages 295–302, 2007.

[128] E. C. Tolman. Cognitive maps in rats and men. *Psychological review*, 55(4), 1948.

[129] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow. The anatomy of the Facebook social graph. *Computing Research Repository (CoRR)*, 2011.

[130] M. Utiyama and H. Isahara. A statistical model for domain-independent text segmentation. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 499–506, 2001.

[131] T. W. Valente, K. Coronges, C. Lakon, and E. Costenbader. How correlated are network centrality measures? *Connections*, 8:16–26, 2008.

[132] X. F. Wang and G. Chen. Complex networks: small-world, scale-free and beyond. *IEEE Circuits and Systems Magazine*, 3(1):6–20, 2003.

[133] D. Watts and S. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998.

[134] M. Wertheimer. Untersuchungen zur lehre von der gestalt. ii. *Psychologische Forschung*, 4(1):301–350, 1923.

[135] K.-F. Wong, M. Wu, and W. Li. Extractive summarization using supervised and semi-supervised learning. In *Proceedings of the 22nd International Conference on Computational Linguistics*, pages 985–992, 2008.

[136] S. K. M. Wong, W. Ziarko, and P. C. N. Wong. Generalized vector spaces model in information retrieval. In *Proceedings of the 8th ACM Conference on Research and Development in Information Retrieval*, pages 18–25, 1985.

[137] H. C. Wu, R. W. P. Luk, K.-F. Wong, and K.-L. Kwok. Interpreting tf-idf term weights as making relevance decisions. *ACM Transactions on Information Systems*, 26(3), 2008.

[138] J. Xu and H. Li. AdaRank: A boosting algorithm for information retrieval. In *Proceedings of the 30th ACM Conference on Research and Development in Information Retrieval*, 2007.

[139] J. Yamron, I. Carp, L. Gillick, S. Lowe, and P. Van Mulbregt. A Hidden Markov model approach to text segmentation and event tracking. In *Proceedings of the Conference on Acoustics, Speech and Signal Processing*, pages 333–336, 1998.

[140] D. Yarowsky and R. Wicentowski. Minimally supervised morphological analysis by multimodal alignment. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 207–216, 2000.

[141] S.-C. Zhu. Embedding gestalt laws in Markov random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(11):1170–1187, 1999.